

日 本 国 特 許 庁
JAPAN PATENT OFFICE

07.10.03

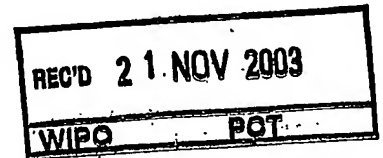
別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日
Date of Application: 2002年10月15日

出 願 番 号
Application Number: 特願2002-300073
[ST. 10/C]: [JP2002-300073]

出 願 人
Applicant(s): 株式会社ルネサステクノロジ

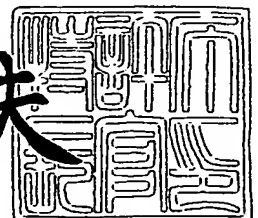


**PRIORITY
DOCUMENT**
SUBMITTED OR TRANSMITTED IN
COMPLIANCE WITH RULE 17.1(a) OR (b)

2003年11月 7日

特許庁長官
Commissioner,
Japan Patent Office

今 井 康 夫



【書類名】 特許願

【整理番号】 H02014091

【提出日】 平成14年10月15日

【あて先】 特許庁長官殿

【国際特許分類】 G06F 19/00

【発明者】

 【住所又は居所】 東京都小平市上水本町五丁目20番1号 株式会社日立
 製作所 半導体グループ内

 【氏名】 谷本 匡亮

【発明者】

 【住所又は居所】 東京都小平市上水本町五丁目20番1号 株式会社日立
 製作所 半導体グループ内

 【氏名】 鎌田 丈良夫

【特許出願人】

 【識別番号】 000005108

 【氏名又は名称】 株式会社日立製作所

【代理人】

 【識別番号】 100089071

 【弁理士】

 【氏名又は名称】 玉村 静世

 【電話番号】 03-5217-3960

【手数料の表示】

 【予納台帳番号】 011040

 【納付金額】 21,000円

【提出物件の目録】

 【物件名】 明細書 1

 【物件名】 図面 1

 【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 コンパイラ及び論理回路の設計方法

【特許請求の範囲】

【請求項 1】 所定のプログラム言語を流用して記述された第 1 プログラム記述を回路記述に変換可能なコンパイラであって、

前記第 1 プログラム記述は、サイクル精度で回路動作を特定可能とするレジスタ代入文とクロック境界記述を含み、

前記回路記述は、前記第 1 プログラム記述が特定する回路動作を実現するハードウェアを所定のハードウェア記述言語で特定することを特徴とするコンパイラ。

【請求項 2】 所定のプログラム言語を流用して記述された第 1 プログラム記述を所定のプログラム言語を用いた第 2 プログラム記述に変換可能なコンパイラであって、

前記第 1 プログラム記述は、サイクル精度で回路動作を特定可能とするレジスタ代入文とクロック境界記述を含み、

前記第 2 プログラム記述は、前のサイクルの状態を参照可能にする為に前記レジスタ代入文を変形した変形代入文と、前記クロック境界記述に対応して前記変形代入文の変数をサイクル変化に伴うレジスタの変化に対応させるレジスタ代入記述挿入文とを含むことを特徴とするコンパイラ。

【請求項 3】 所定のプログラム言語を流用して記述された第 1 プログラム記述を、所定のプログラム言語を用いた第 2 プログラム記述と回路記述に変換可能なコンパイラであって、

前記第 1 プログラム記述は、サイクル精度で回路動作を特定可能とするレジスタ代入文とクロック境界記述を含み、

前記第 2 プログラム記述は、前のサイクルの状態を参照可能にする為に前記レジスタ代入文を変形した変形代入文と、前記クロック境界記述に対応して前記変形代入文の変数をサイクル変化に伴うレジスタの変化に対応させるレジスタ代入記述挿入文とを含み、

前記回路記述は、前記第 2 プログラム記述で定義されるハードウェアを所定の

ハードウェア記述言語で特定することを特徴とするコンパイラ。

【請求項 4】 前記所定のプログラム言語は C 言語であることを特徴とする請求項 1 乃至 3 の何れか 1 項記載のコンパイラ。

【請求項 5】 前記ハードウェア記述言語は R T L レベルの記述言語であることを特徴とする請求項 1 又は 3 記載のコンパイラ。

【請求項 6】 タイミング仕様に基づいて回路動作を定義するために、所定のプログラム言語を流用して記述され、サイクル精度で回路動作を特定可能とするレジスタ代入文とクロック境界記述を含む第 1 プログラム記述を入力する第 1 処理と、

前記第 1 プログラム記述に基づいて前記タイミング仕様を満足する回路情報を生成する第 2 処理と、を含むことを特徴とする論理回路の設計方法。

【請求項 7】 前記第 2 処理は、前記第 1 プログラム記述を変換して、レジスタ代入文が入力変数と出力変数を用いて変形されると共に前記クロック境界記述に対応させて前記入力変数を出力変数に代入する記述を含む第 2 プログラム記述を前記回路情報として生成する処理を含むことを特徴とする請求項 6 記載の論理回路の設計方法。

【請求項 8】 前記第 2 処理は、前記第 2 プログラム記述を変換して、前記タイミング仕様を満足するハードウェアを所定のハードウェア記述言語で特定するための回路記述を更に別の前記回路情報として生成する処理を含むことを特徴とする請求項 7 記載の論理回路の設計方法。

【請求項 9】 前記プログラム言語は C 言語であることを特徴とする請求項 8 記載の論理回路の設計方法。

【請求項 10】 前記第 2 プログラム記述を用いて設計対象回路のシミュレーションを行う第 3 処理を更に含むことを特徴とする請求項 9 記載の論理回路の設計方法。

【請求項 11】 前記第 2 処理は、前記第 1 プログラム記述を変換して、前記レジスタ代入文が入力変数と出力変数を用いて変形された記述を含む第 2 プログラム記述を前記回路情報として生成する処理を含むことを特徴とする請求項 6 記載の論理回路の設計方法。

【請求項 1 2】 前記第 2 処理は、前記第 2 プログラム記述を変換して、前記クロック境界記述に対応させて前記入力変数を出力変数に代入する記述を含み、所定のプログラム言語で記述されてコンピュータで実行可能な第 3 プログラム記述を、前記回路情報として生成する処理を含むことを特徴とする請求項 1 1 記載の論理回路の設計方法。

【請求項 1 3】 前記第 3 プログラム記述を用いて設計対象回路のシミュレーションを行う第 3 処理を更に含むことを特徴とする請求項 1 2 記載の論理回路の設計方法。

【請求項 1 4】 タイミング仕様に基づいて回路動作を定義するために、所定のプログラム言語を流用して記述され、サイクル精度で回路動作を特定可能とするレジスタ代入文とクロック境界記述を含む第 1 プログラム記述を入力する入力処理と、

前記レジスタ代入文が入力変数と出力変数を用いて変形されると共に前記クロック境界記述に対応させて前記入力変数を出力変数に代入する記述を含み、前記所定のプログラム言語で記述された第 2 プログラム記述を生成する変換処理と、を含むことを特徴とする論理回路の設計方法。

【請求項 1 5】 前記変換処理は、第 1 プログラム記述に基づいて C F G を生成する過程で、前記 C F G に前記クロック境界記述に対応してクロック境界ノードを設定し、前記クロック境界ノードの後に、前記レジスタ代入記述を挿入することを特徴とする請求項 1 4 記載の論理回路の設計方法。

【請求項 1 6】 第 2 プログラム記述に対してその C F G を利用しながらステート遷移毎の変数表を作成しながらコード最適化を行う最適化処理を更に含むことを特徴とする請求項 1 5 記載の論理回路の設計方法。

【請求項 1 7】 前記変数表においてステート間で変数に変化のない部分を前置保持を要する部分として抽出し、抽出された部分に、出力変数に入力変数を代入する記述を追加する前置保持処理を更に含むことを特徴とする請求項 1 6 記載の論理回路の設計方法。

【請求項 1 8】 前記前置保持処理を経た変数表の各ステート遷移毎の変数と引数に基づいてステートマシンを構成するコードの抽出を行う抽出処理を更に

含むことを特徴とする請求項 17 記載の論理回路の設計方法。

【請求項 19】 前記抽出処理で抽出されたステートマシン構成コードと第 2 プログラム記述を参照しながら、前記回路仕様を満足する回路のハードウェアを所定のハードウェア記述言語で記述する処理を更に含むことを特徴とする請求項 18 記載の論理回路の設計方法。

【請求項 20】 前記第 1 プログラム記述に対して 0 サイクルで実行されるループが存在するか否かが判定され、存在しないと判別されたときに前記回路仕様を満足する回路のハードウェアを所定のハードウェア記述言語で記述する処理を行うことを特徴とする請求項 14 記載の論理回路の設計方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明はプログラム記述からシミュレーション用のプログラム記述又はハードウェアを特定する回路記述を自動生成する技術に関し、例えばパイプライン動作される論理回路、例えば CPU (Central Processing Unit) 等の論理回路の設計に適用して有効な技術に関する。

【0002】

【従来の技術】

プログラム言語を用いてデジタル回路の回路記述を生成する技術がある。特許文献 1 に記載の技術では、レジスタを示す変数と、レジスタの入力を示す変数とに分け、モジュール部での処理の後に第 2 の変数から第 1 の変数に一括して代入する一括代入部を設けている。特許文献 2 には、汎用プログラム言語で回路動作を記述したプログラムの中から、順次制御する部分を特定処理部で特定し、その後、変換処理部で、前記順次制御する部分の記述を、ステートマシンとして動作するように汎用プログラム言語を用いて変換し、その変換後のプログラムを取得し、続いて、プログラム生成処理部で、前記変換後のプログラムの中から並行動作する部分を抽出し、この抽出部分の全てをアクセスするプログラムを生成する、というものである。

【0003】

【特許文献 1】

特開 2002-49652 号公報

【特許文献 2】

特開平 10-149382 号公報

【0004】

【発明が解決しようとする課題】

特許文献 1 によれば、①回路動作を示すモジュール、②レジスタ代入を行う一括代入部、③クロック同期で繰り返すループ部の 3 つの構成からなっており、特に③内で①の実行後に②を実行する事を特徴としている。しかしながら、①がクロック境界を含まず、必ず③に含まれる構成となるため、複数サイクルにまたがる回路動作を記述する為には、回路動作をクロック境界で分割する必要がある。例えば、ある条件が成立したときは、前サイクルで実行した回路動作の途中から回路動作を行うという記述を行わなければならないが、そのような記述を行うのは困難である。特に、ストール動作を伴うパイプライン動作を行う回路を特許文献 1 に示す方法で記述すると、煩雑な作業を伴い、かつプログラム記述が複雑なものになる虞のあることが本発明者によって見出された。

【0005】

特許文献 2 によれば、①汎用言語で記述したプログラムから順次処理部を識別し、ステートマシンを表す汎用プログラム記述に変換、②関数レベルでの並列性の抽出、③ハード化するプログラムとそれを制御するソフトプログラムの結合の自動化、④順次処理部内でハード化する際にフリップフロップやラッチを必要とする部分を識別して HDL に変換、の 4 つを特徴点がある。しかしながら、クロック境界を明示的に与えられる手段がなく、サイクル精度での記述を直接行う事が出来ない。特許文献 2 の実施例によれば、クロック境界は関数から関数への間であり、例えばある条件が成立したときは、前サイクルで実行した回路動作の途中から回路動作を行うという記述を行うのが困難である。特に、ストール動作を伴うパイプライン動作を行う回路を特許文献 2 に示す方法で記述することは可能であるが、煩雑な作業を伴い、かつプログラム記述が複雑になる虞のあることが本発明者によって見出された。

【0006】

本発明の目的は、クロック境界を明示的に記述したプログラム記述からハードウェア記述を自動生成することができるコンパイラを提供することにある。

【0007】

本発明の別の目的は、ストール動作を伴うパイプライン動作が可能な回路のプログラム記述又は回路記述を容易に得る事ができるコンパイラを提供することにある。

【0008】

本発明の更に別の目的は、ストール動作を伴うパイプライン動作が可能な回路の設計を行うことができる論理回路の設計方法を提供することにある。

【0009】

本発明の前記並びにその他の目的と新規な特徴は本明細書の記述及び添付図面から明らかになるであろう。

【0010】

【課題を解決するための手段】

本願において開示される発明のうち代表的なものの概要を簡単に説明すれば下記の通りである。

【0011】

〔1〕本発明の概要を全体的に説明する。即ち、クロック境界（記述子\$）及びレジスタ代入文（演算子=\$を挟む記述）によりステートメントレベルでの並列動作の記述をサイクル精度で記述可能な擬似C記述（1）を入力とし、レジスタ代入文の識別を行い（S2）、実行可能なC記述（3）を生成する（S3およびS4）と共に、状態数削減を行ったステートマシンを抽出し、0サイクルで実行されるループが存在するか否かを判定し（S5）、もしなければ、論理合成可能な回路記述（4）を生成する（S6）。

【0012】

上記より、クロック境界を明示的にC記述内に挿入した擬似C記述を入力し、レジスタ代入文によるステートメントレベルでの並列記述を可能にした擬似C記述を入力するから、ストール動作を伴うパイプライン動作が表現可能である。

【0013】

擬似C記述から一般のCコンパイラによるコンパイルが可能なC記述を出力することができる。状態（ステート）数削減を行うので、記述で与えたクロック境界の数+1以下のステート数のステートマシンを伴う回路記述を出力することができる。

【0014】

ステートマシンを意識する事なくプログラム・レベルで機能設計を行う事ができるため、記述量が低減され、開発期間の短縮のみならず品質向上にも寄与する。

【0015】

また、一般のクロック境界を指定しないプログラム・レベルでの記述では表現できない、バス・インターフェース回路や調停回路の記述が可能となる。特に、レジスタ代入が記述可能である為、ステートメントレベルでの並列性を考慮した記述を行う事が可能であり、ストール動作を伴うパイプライン動作のような複雑な回路動作をC記述よりも少ないコード量で容易に記述可能である。

【0016】

また、一般のCコンパイラでコンパイル可能なC記述へ変換する為、高速なシミュレーションが可能となり、機能検証工数の大幅な低減が可能となる。従って、機能設計における論理設計、論理検証の双方の大幅な工数削減が可能となる。

【0017】

クロック境界を指定したプログラム記述からミーリー（Mealy）型のステートマシンが生成可能であるので、プログラム・レベルでのモデル検査を行う事が可能である。

【0018】

高位合成ツールが不得意とする、サイクル精度を要求される例えば、キャッシュ・コントローラやDMAコントローラの開発に適用可能であり、設計期間の短縮に大きく寄与する。

【0019】

〔2〕本発明に係るコンパイラの第1形態では、コンパイラは、所定のプログ

ラム言語を流用して記述された第1プログラム記述(1)を回路記述(4)に変換可能であって、前記第1プログラム記述は、サイクル精度で回路動作を特定可能とするレジスタ代入文(演算子=\$を挟む記述)とクロック境界記述(\$)を含み、前記回路記述は、前記第1プログラム記述が特定する回路動作を実現するハードウェアを所定のハードウェア記述言語で特定する。

【0020】

本発明に係るコンパイラの第2形態では、コンパイラは、所定のプログラム言語を流用して記述された第1プログラム記述を所定のプログラム言語を用いた第2プログラム記述(3)に変換可能であり、前記第1プログラム記述は、サイクル精度で回路動作を特定可能とするレジスタ代入文(演算子=\$を挟む記述)とクロック境界記述(\$)を含む。前記第2プログラム記述は、前のサイクルの状態を参照可能に前記レジスタ代入文を変形した変形代入文(13)と、前記クロック境界記述に対応して前記変形代入文の変数をサイクル変化に伴うレジスタの変化に対応させるレジスタ代入記述挿入文(12)とを含む。

【0021】

本発明に係るコンパイラの第3形態では、コンパイラは、所定のプログラム言語を流用して記述された第1プログラム記述(1)を、所定のプログラム言語を用いた第2プログラム記述(3)と回路記述(4)に変換可能である。前記第1プログラム記述は、サイクル精度で回路動作を特定可能とするレジスタ代入文とクロック境界記述を含む。前記第2プログラム記述は、前のサイクルの状態を参照可能に前記レジスタ代入文を変形した変形代入文と、前記クロック境界記述に対応して前記変形代入文の変数をサイクル変化に伴うレジスタの変化に対応させるレジスタ代入記述とを含む。前記回路記述は、前記第2プログラム記述で定義されるハードウェアを所定のハードウェア記述言語で特定する。

【0022】

前記所定のプログラム言語は例えばC言語である。前記ハードウェア記述言語は例えばRTLレベルの記述言語である。

【0023】

〔3〕本発明に係る論理回路の設計方法の第1形態では、タイミング仕様に基

づいて回路動作を定義するために、所定のプログラム言語を流用して記述され、サイクル精度で回路動作を特定可能とするレジスタ代入文（演算子＝\$を挟む記述）とクロック境界記述（\$）を含む第1プログラム記述（1）を入力する第1処理（S1）と、前記第1プログラム記述に基づいて前記タイミング仕様を満足する回路情報を生成する第2処理と、を含む。

【0024】

前記第2処理は、前記第1プログラム記述を変換して、前記レジスタ代入文が変数と出力変数を用いて変形される（S2）と共に前記クロック境界記述に対応させて前記変数を出力変数に代入する（S4）記述（13, 12）を含む第2プログラム記述（3）を、前記回路情報として生成する処理を含んでよい。

【0025】

前記第2処理は、前記第2プログラム記述に基づいて前記タイミング仕様を満足するハードウェアを所定のハードウェア記述言語で特定するための回路記述（4）を更に別の前記回路情報として生成する処理を含んでよい。

【0026】

前記第2プログラム記述を用いて設計対象回路のシミュレーションを行う第3処理を更に含んでもよい。

【0027】

上記第2処理に関し、前記レジスタ代入文が変数と出力変数を用いて変形される（S2）記述（13）を含む第2プログラム記述（5）と、前記クロック境界記述に対応させて前記変数を出力変数に代入する（S4）記述（12）を含む第3プログラム記述（3）とを、分けて把握することも可能である。このとき、第3処理によりシミュレーションは第3プログラム記述に基づいて行うことになる。

【0028】

〔4〕本発明に係る論理回路の設計方法の第2形態では、タイミング仕様に基づいて回路動作を定義するために、所定のプログラム言語を流用して記述され、サイクル精度で回路動作を特定可能とするレジスタ代入文とクロック境界記述を

含む第1プログラム記述を入力する入力処理(S1)と、前記レジスタ代入文が入力変数と出力変数を用いて変形される(S2)と共に前記クロック境界記述に対応させて前記入力変数を出力変数に代入する(S4)記述(13, 12)を含み、前記所定のプログラム言語で記述された第2プログラム記述を生成する変換処理とを含む。

【0029】

前記変換処理は、第1プログラム記述に基づいてCFGを生成する過程で、前記CFGに前記クロック境界記述に対応してクロック境界ノードを設定し、前記クロック境界ノードの後に、前記レジスタ代入記述を挿入する処理であってよい。

【0030】

第2プログラム記述に対してそのCFGを利用しながら状態遷移毎の変数表を作成しながらコード最適化を行う最適化処理を更に含んでもよい。

【0031】

前記変数表において状態間で変数に変化のない部分を前置保持を要する部分として抽出し、抽出された部分に、出力変数に入力変数を代入する代入記述を追加する前置保持処理を更に含んでもよい。

【0032】

前記前置保持処理を経た変数表の各状態遷移毎の変数と引数に基づいてステートマシンを構成するコードの抽出を行う抽出処理を更に含んでもよい。

【0033】

前記抽出処理で抽出されたステートマシン構成コードと第2プログラム記述を参照しながら、前記回路仕様を満足する回路のハードウェアを所定のハードウェア記述言語で記述する回路記述を生成する処理を更に含んでもよい。

【0034】

前記第1プログラム記述に対して0サイクルで実行されるループが存在するかが判定され、存在しないと判別されたときに前記変換処理が行なわれる。

【0035】

【発明の実施の形態】

《設計方法の概略》

図1には本発明に係る論理回路の設計方法が例示される。同図に示される設計方法は、擬似C記述（擬似Cプログラム）1の作成、擬似Cプログラム1に対するコンパイル処理2に大別される。コンパイル処理2では、擬似Cプログラム1を、レジスタ代入記述を変形代入文とした擬似Cプログラム（5に格納）、および実行可能なC記述（Cプログラム）3に変換し、また、そのCプログラム3をRTL（Register Transfer Level）などのHDL（Hardware Description Language）記述4に変換する。

【0036】

前記擬似Cプログラム1は、サイクル精度で回路動作を特定可能とするクロック境界記述（単にクロック境界とも記す）及びレジスタ代入文を含み、ステートメントレベルでの並列記述を可能にしたプログラムである。擬似C記述とは、前記クロック境界及びレジスタ代入文が定義されていない所謂ネイティブのC言語記述とは相違するという意味で用いられている。プログラム言語としてC言語以外の高級言語をベースとすることを妨げるものではない。

【0037】

コンパイル処理2は、図示を省略するコンピュータ装置がコンパイラを実行し、擬似Cプログラム1を読み込んで行なわれる。先ず擬似Cプログラム1が読み込まれる（S1）。読み込まれた擬似Cプログラム1に対しては、レジスタ代入文の識別が行なわれ、識別されたレジスタ代入文を、前のサイクルの状態を参照可能に変形し、換言すれば、入力変数と出力変数を用いて変形する（S2）。変形されたレジスタ代入文を変形代入文とも称する。レジスタ代入文が変形代入文に変形された擬似Cプログラムはレジスタ情報記憶部5に格納される。レジスタ代入文が変形代入文に変形された擬似Cプログラムは前記レジスタ情報記憶部5から取り出されて、そのコントロール・フロー・グラフ（以下CFGと記す）が生成される（S3）。生成されたCFGは中間表現記憶部6に格納される。前記中間表現記憶部6に格納されたCFG及び前記レジスタ情報記憶部5に格納された擬似Cプログラムは、実行可能なC記述プログラムに変換される（S4）。例えば、前記クロック境界記述に対応して前記変形代入文の変数をサイクル変化に

伴うレジスタの変化に対応させるレジスタ代入記述挿入文が挿入される。換言すれば、クロック境界記述に対応させて前記変形代入文の入力変数を出力変数に代入するレジスタ代入記述挿入文が挿入される。

【0038】

前記擬似Cプログラム5等に基づいてHDL記述4を得る場合、先ずそれらを入力してステートマシンの生成が行なわれる(S5)。ステートマシン生成(S5)は、ステート数削減処理(S5A)、コードの最適化(S5B)、HDL記述に則するための前置保持解析(S5C)、及びステートマシン抽出(S5D)に大別される。ステート数削減処理(S5A)とコードの最適化(S5B)は最適化処理の範疇に属する処理と把握してもよい。コードの最適化(S5B)の段階では、0サイクルで実行されるループが存在するか否かを判定し、もしなければ、HDL記述に則するための前置保持解析(S5C)、及びステートマシン抽出(S5D)が行われる。前記C記述プログラムを得るときには、例えばクロック境界ノードに前記レジスタ代入記述挿入文を挿入すればよかったが、HDL記述を得るときはクロック境界でレジスタ値が変化しない場合にもそれを明示的に記述しておくことが必要とされる。そのために、前置保持解析(S5C)が行なわれる。生成されたステートマシンはステート遷移毎の変数表に基づいて生成される。生成されたステートマシンはステートマシン記憶部7に保持される。保持されたステートマシン等に基づいてHDL記述4が生成される(S6)。

【0039】

HDL記述4は論理合成ツールを利用することによって論理回路図データに変換可能にされる。前記C記述3は前記論理合成される論理回路のシミュレーションなどに利用される。

【0040】

以下に、上記擬似Cプログラムとそのコンパイル処理を詳細に説明する。以下の詳細説明は図2の回路に図3の仕様を満足させる回路の設計を一例とする。

【0041】

《設計対象回路》

図2には図1の設計方法を適用して設計すべき回路例が示される。設計対象回

路 10 はストール動作を伴うパイプライン加算回路である。その動作仕様は以下の通りである。

(1) 入力信号valid_aが立ち上がると、信号レベルのハイレベルとなったサイクルの入力信号aの値を取り込む。ここではvalid_aが立ち上がり変化を問題にする。

(2) 入力信号valid_aの立ち上がりの次サイクル以降で、入力信号valid_bの信号レベルがハイレベルとなると、そのサイクルでの入力信号bの値を取り込む。入力信号valid_bに対してはレベル検出だけで充分とされ、エッジ変化の検出は不要とされる。

(3) 上記 (1) (2) の動作でaとbが取り込まれたなら、その次サイクルでaとbの加算結果を出力信号outにより送出し、その同一サイクルに出力信号valid_outの信号レベルをハイレベルとし、次サイクルで出力信号valid_outの信号レベルをロウレベルとする。

(4) 出力信号outは (1) (2) (3) の動作での新たな加算結果が代入されない限り、同じ値を出力する。

(5) 出力信号valid_outは (1) (2) (3) の動作で出力信号outへ新たな加算結果が代入されたサイクルのみ信号レベルがハイとなり、それ以外はロウレベルを出力する。

【0042】

図3には図2の回路動作仕様を示すタイミングチャートである。同図において、出力データ送出と入力データ取り込みが同一サイクルで行われており、パイプライン動作となっている。例えばa2の入力とa1+b1の出力が並列化されている。また、入力信号valid_aの立ち上がりの次サイクル以降で入力信号valid_bの値が1となった次のサイクルで出力データ送出が行われる為、ストール動作を伴うパイプライン動作となっている。例えばb1の取込み後におけるb2の取込みは2サイクル待たされている。

【0043】

《擬似Cプログラム》

図4には前記設計対象回路10の擬似Cプログラムが例示される。図4に記述

において 11 は、設計対象回路 10 の回路動作を記述した回路動作記述部である。同図に示される擬似 C プログラムの記述は以下の通りである。即ち、

1 行目: C 言語でのライブラリ呼び出し、

2 ~ 7 行目: 関数 pipeline のプロトタイプ宣言部、

8 ~ 14 行目: main 関数部、

9 ~ 10 行目: main 関数のローカル変数宣言部。出力信号はポインタ型で宣言、

11 ~ 12 行目: main 関数のローカル変数の初期化 (出力信号のみ初期化、特に出力信号に対して RTL への変換時にレジスタが推定される場合ここで指定した初期値がリセット値となる)、

15 ~ 36 行目: pipeline 関数部、

18 ~ 20 行目: pipeline 関数のローカル変数宣言部 (特にローカル変数に対して RTL への変換時にレジスタが推定される場合ここで指定した初期値がリセット値となる)、

21 ~ 35 行目: 回路動作記述部 11、である。

【0044】

回路動作記述部 11 の詳細は以下の通りである。即ち、

21、35 行目: 無限ループにより回路を表現、

22 行目: 入力変数 valid_a のローカル変数 valid_a_tmp へのレジスタ代入文 (ここで、0x0001 & valid_a により、入力変数 valid_a の有効ビット幅が 1 ビットである事を指定している)、

23 行目: valid_a が 1 'b1 で valid_a_tmp が 1' b0 であるか否かの判定文 (即ち、valid_a が立ち上がりであるか否かの判定文。特に、0x0001 & valid_a_tmp により、ローカル変数 valid_a_tmp の有効ビット幅が 1 ビットである事を指定している)、

24 行目: 入力信号 a のローカル変数 a_tmp への代入文 (特に、0x7FFF & a により、入力変数 a の有効ビット幅が 15 ビットである事を指定している)、

25 行目: クロック境界、

26 行目: goto ラベル、

2 7 ~ 2 8 行目：入力変数valid_bが1 'b1であれば、ローカル変数b_tmpに変数bを代入し、そうでなければクロック境界を1つまたいでラベルLへ分岐する事を表している（特に、0x0001&valid_bにより、入力変数bの有効ビット幅が1ビットである事を、0x7FFF&bにより、入力変数bの有効ビット幅が15ビットである事を表している）、

2 9 行目：ローカル変数a_tmpとローカル変数b_tmpの和の出力変数outへのレジスタ代入文、

3 0 行目：定数0x0001の出力変数valid_outへのレジスタ代入文、

3 1 行目：2 3 行目のif文の判定が成立しなかった場合の分岐。即ち、valid_aが立ち上がりでなかった場合の分岐を表す、

3 2 行目：クロック境界、

3 3 行目：定数0x0000の出力信号valid_outへのレジスタ代入文、である。

【 0 0 4 5 】

上記記号“\$”はクロック境界記述を意味し、記号“=\$”レジスタ代入を意味する。それらはC言語の汎用的な記述子及び演算子ではない。これを用いた擬似Cプログラムは、その意味においてC言語を流用したプログラム記述と言うことができる。

【 0 0 4 6 】

上記回路動作記述部11より明らかなように、クロック境界記述及びレジスタ代入文によりステートメントレベルで並列動作をサイクル精度で簡単に記述可能になる。サイクル精度とは、クロックサイクルとの同期が意図される、ということである。

【 0 0 4 7 】

図4の回路動作記述部11の記述内容について説明する。入力変数valid_aをローカル変数valid_a_tmpに代入する事で、if文によるvalid_aの立ち上がり判定を行い、もし立ち上がりであった場合は、ローカル変数a_tmpに入力信号aを取り込み、次のサイクルで入力信号valid_bが1 'b1であるか否かを判定する。もしそうなら入力信号bの値をローカル変数b_tmpに代入し、そうでなければ次のサイクルでもう一度入力信号valid_bが1 'b1であるか否かを判定する。これを入力信号

valid_bが1 'b1となるまで繰り返す。この動作がストール動作に対応している。さて、ローカル変数a_tmpとb_tmpの和は取り込んだaとbの値の和を表しており、それを出力変数outへレジスタ代入し、同時に1' b1を出力信号valid_outへレジスタ代入している。これにより、入力信号aとbを取り込んだ1サイクル後での加算結果とvalid_out信号が1 'b1である事を表現している。if文によるvalid_aの立ち上がり判定を行い、立ち上がりでない場合は、1サイクル後に1 'b0をvalid_outへレジスタ代入している。valid_aの立ち上がりは高々2サイクルに1回しか起こり得ないので、変数outへの新たな代入が29行目で行われた時のみvalid_outが1 'b1となり、それ以外の場合は、1' b0となる。

【0048】

図4の第22行におけるレジスタ代入文は、サイクル精度で動作を特定するのに順序回路としてのレジスタを想定しており、左辺 (valid_a_tmp) はレジスタの出力、即ち前サイクルの値を保持している変数として把握可能である。レジスタ代入文の右辺 (0x0001&valid_a) は現時点のレジスタ入力として把握可能である。また、図4の第29行目及び第30行目に記載のレジスタ代入文に関しては、その後の第32行におけるクロック境界記述でクロックが消費されるようになっているが、図2及び図3の回路仕様ではその次サイクルでoutを出力するとあり、結果として、out、valid_outに関しては必然的にサイクル精度の記述が必要になるため、それらの記述にはレジスタ代入文が用いられている。

【0049】

《レジスタ代入文識別》

次にレジスタ代入文識別処理S2について説明する。前記レジスタ代入文識別処理部では、代入文であって、=と右辺の間に\$が付加された文を識別し、回路動作記述部11内のレジスタ代入文、レジスタ代入文の左辺の変数の型と初期値を記憶し、識別したレジスタ代入文signal_latched = \$ signal;を
 signal_latched_i = signal;
 signal_latched = signal_latched_o;
 の記述に変更する。signal_latched_iは現時点の入力が与えられる入力変数、signal_latched_oは1サイクル前の出力が当てられる出力変数として把握すること

が可能である。変数宣言部に変更により生じた新たな変数

signal_latched_i, signal_latched_o

を先に記憶しておいた変数の型と初期値を参照して追加する。例えば、

```
unsigned char signal_latched = 0x01;
```

の場合は、

```
unsigned char signal_latched_o = 0x01;
```

```
unsigned char signal_latched_i;
```

を追加する。特に、レジスタ代入の左辺の変数が、ポインタ型の場合（記号*が付されている）は、そのポインタ型を用いて変数宣言を行う。例えば、

```
unsigned char *signal_latched;
```

の場合は、

```
unsigned char signal_latched_o = 0x01;
```

```
unsigned char signal_latched_i;
```

を追加する。特に、追加対象となった変数に対して、同じ型で初期値を0としたフラグ変数も追加予定として、記憶する。この例の場合、

```
unsigned char flg_signal_latched = 0x00;
```

を追加予定変数として記憶する。尚、上記変更を行った記述も記憶する。また、変数の初期値はHDL変換時に該変数へのレジスタ推定が行われた場合、リセット時の値として用いる。

【0050】

図5にはレジスタ代入文識別処理S2によって得られる結果が例示される。図4の擬似Cプログラム対して追加変数宣言の記述と変形代入文（レジスタ代入文書き換え）13の記述が変更されている。

【0051】

《CFG生成》

次にCFG生成処理について説明する。CFGとは、一般に各関数内部において制御の流れを示すグラフを意味する。

【0052】

CFG生成処理では、回路動作記述部11を読み込んで、CFGの作成を行う

。特にwhileやfor等のループ及びifやcase等の条件分岐、goto文によるラベルへのラベル分岐を識別する為のノードを持つC F Gの作成を行う。要するに、whileやfor等のループ及びifやcase等の条件分岐、goto文によるラベルへのラベル分岐をノードに持つC F Gを作成する。各文をプログラムの終了迄読み込み、以下の手順1)～7)でノードを作成しながらプログラムの流れに沿って、ノード間の接続を有向辺(向きが付いている辺)で接続する事でC F Gを作成する。図6から図21には手順1)～7)によるC F Gの作成過程が順を追って示される。各図にはループ文スタック、分岐文スタック、生成途中のC F Gが示される。

【0053】

1) ループの開始であれば、ループ文スタックにその行番号とwhileやfor等のループを表す終端記号を登録し、ループ開始ノード(N D s)を作成し、行番号と終端記号をノードに付加する。また、forやwhileループ終了条件があれば、その条件を適当な記号に代入し、出力枝に付加し、付加した条件を割り当てた記号との対で記憶する。

【0054】

2) ループの終了であれば、ループ文スタックから先頭にある情報を取り去り、ループ終了を表すループ終了ノードを作成し、行番号と“end of 終端記号”をノードに付加する。但し、continueやbreakはループの終了としては扱わない。また、do-whileループ終了条件があれば、その条件を適当な記号に代入し、出力枝に付加し、付加した条件を割り当てた記号との対で記憶する。

【0055】

3) 条件分岐の開始であれば、分岐文スタックにその行番号とifやcase等の分岐を表す終端記号を登録し、条件分岐開始ノードを作成し、行番号と終端記号をノードに付加する。また、分岐条件を適当な記号に代入し、出力枝に付加し、付加した条件を割り当てた記号との対で記憶する。

【0056】

4) 条件分岐の終了であれば、分岐文スタックから先頭にある情報を取り去り、条件分岐終了を表す条件分岐終了ノードを作成し、行番号と“end of 終端記号”をノードに付加する。

【0057】

5) ラベルであれば、ラベルを表すラベルノードを作成し、行番号とラベル記号をノードに付加する。

【0058】

6) クロック境界であれば、クロック境界ノードを作成し、行番号と\$をノードに付加する。

【0059】

7) 上記以外であれば、行番号と文を付加したノードを作成し、1)～6)の何れかに出会うまでノードをマージする。

【0060】

上記手順によるCFGが作成されるが、以下の説明では、その説明を簡単化するために、図22に例示されるように、クロック境界や分岐の始点・終点、及びループの始点・終点の情報を付加していないCFGを用いて説明を行う。特に、クロック境界ノードのみ黒丸で、それ以外のループ、条件分岐、ラベル分岐ノードを白丸で表現する。

【0061】

《C記述生成》

前記C記述生成処理S4について説明する。C記述生成処理S4では、前記レジスタ代入文識別処理で追加予定変数として記憶しておいた変数で、レジスタ代入部識別処理で変更した部分（変形代入文）の直下に対応するフラグ変数に1を代入する文の挿入を行い、レジスタ代入文の左辺の変数への代入文でレジスタ代入文でない代入文の直下に対応するフラグ変数に0を代入する文の挿入を行う。また同時に、ローカル変数宣言部に、レジスタ代入文識別部で記憶しておいた変数宣言を追加する。図23ではflg_valid_a_tmp=1、flg_valid_out=1のフラグが挿入されている。

【0062】

次にレジスタ代入記述挿入文が決定される。前記レジスタ代入文識別処理S2において、識別されたレジスタ代入文の右辺の変数全てに対して、レジスタ代入記述挿入文が作成される。即ち、

レジスタ代入文：

```
signal_latched = $ signal;
```

変更後の記述：

```
signal_latched_i = signal;
```

```
signal_latched = signal_latched_o;
```

追加された変数：

```
signal_latched_i, signal_latched_o, flg_signal_latched
```

とされている場合、下記記述

```
signal_latched_o = signal_latched_i;
```

```
if (flg_signal_latched==1) signal_latched = signal_latched_o;
```

を作成する。これをレジスタ代入文識別処理で識別したレジスタ代入文の右辺の変数全てに対して作成する。例の場合には、下記記述

```
valid_a_tmp_o = valid_a_tmp_i;
```

```
if (flg_valid_a_tmp==1) valid_a_tmp = valid_a_tmp_o;
```

```
out_o = out_i;
```

```
if (flg_out==1) *out = out_o;
```

```
valid_out_o = valid_out_i;
```

```
if (flg_valid_out==1) *valid_out = valid_out_o;
```

が得られる。

【 0 0 6 3 】

上記レジスタ代入記述挿入文は、図 2 4 に例示されるように、クロック境界ノードの直下に挿入される。図 2 4 においてレジスタ代入記述挿入文には参照符号 1 2 が付されている。このようにして行なわれる C 記述への変換は、各ノードに付加された行番号等の情報を元に、深さ優先探索等のアルゴリズム（D F S）を用いて、C F G を探索する事で挿入文の順番を考慮して行えば良い。尚、適度にコメント文を挿入しても良い。

【 0 0 6 4 】

図 2 5 乃至図 2 7 には上記 C 記述生成処理 S 4 を経て得られる実行可能な変換 C 記述（C プログラム） 3 の全体が例示される。

【0065】

《ステートマシン生成—ステート数削減》

前記ステートマシンの生成処理 S 5 について説明する。ステート数削減処理 S 5 A は例えば第 1 又は第 2 のルールに従って行なわれる。ステート数削減処理の第 1 のルールは図 28 に例示される。即ち、ループ開始・終了ノード、条件分岐開始・終了ノード、ラベル分岐ノードの何れかであって、入力辺が複数あるノードを探索し、その入力辺の内 2 つ以上の入力辺にクロック境界がある場合は、同図に示すグラフ変形を行う。ステート数削減処理の第 2 のルールは図 29 に例示される。即ち、ループ開始・終了ノード、条件分岐開始・終了ノード、ラベル分岐ノードの何れかであって、出力辺が複数あり且つ出力辺に付加された条件が入力信号も出力信号の何れも含まず、2 本以上の出力辺にクロック境界が付加されたノードを探索し、その前段のクロック境界が出力辺のクロック境界を含まない場合、同図に示すグラフ変形を行う。図 30 には図 22 の C F G に対してステート数削減を行った結果が例示される。

【0066】

《ステートマシン生成—コード最適化》

コード最適化処理 S 5 B では前記ステート数削減等の処理を行った C F G に対しては図 31 に例示されるようにステートの割り当てを行う。図 31 に従えば、回路動作部の開始文に対応する C F G 上のノードに初期ステートを割り当て、C F G 上のクロック境界ノードにステートを割り当てる。但し、開始ノードへの入力辺が 1 つしか存在せずクロック境界が付加されている場合は、既に割り当てた初期ステートを削除する。尚、最適化の第 1 ルールにより、初期ステート削除が起こる必要十分条件は、開始ノードへの入力辺が 1 つしか存在せずクロック境界が付加されている事である事に注意することが望ましい。また、得られるステート数は、必ず回路動作部に記述したクロック境界の数 + 1 以下となる事に注意すべきである。

【0067】

ここで、前記コード最適化の処理を、特別に簡素化した別の例を用いて、図 32 乃至図 48 を参照しながら説明する。

【0068】

図32はコード最適化対象とされる擬似Cプログラムを示す。この擬似Cプログラムに基づいて得られたCFGは図33に例示される。図34には図33のCFGに対して状態割り当てが行なわれた状態を例示する。

【0069】

図35から図40までは状態マシン生成のための変数表作成処理の様子が順を追って例示される。変数表の作成は、以下の(1)～(3)の手順で行う。

(1) ローカル変数を取得し、(2) 関数の引数を取得し、(3) 割り当てた状態から状態に到達するまでCFGを下位側に辿って、状態遷移を識別すると共に変数の定義・参照の情報を取得する。この段階で、両端がクロック境界ではないループが発見されると、ゼロサイクル・ループを検出したとして、ユーザに通知し、処理を終了。ゼロサイクル・ループの発生は、生成される回路に組合せ回路からなるループ回路が存在する事を意味しており、ループ回路の存在は生成される回路に重大なミスがあることを意味する。図35には状態ST0からST1への一つの状態遷移におけるローカル変数と引数が例示される。図36には状態ST0からST1への別の状態遷移におけるローカル変数と引数が例示される。図37には状態ST0からST2への状態遷移におけるローカル変数と引数が例示される。図38には状態ST1からST0への状態遷移におけるローカル変数と引数が例示される。図39には状態ST2からST0への状態遷移におけるローカル変数と引数が例示される。図35から図39に示される夫々の状態遷移で得られたローカル変数と引数に基づいて、図40に例示される変数表が生成される。図40の変数表の記述において、def[n]: n行目で変数定義されている事を表し、use@var[m]: m行目で変数varへの代入に用いられている事を表し、pred(cond){...}: 条件condの分岐が成立した場合、{...}が実施される事を表し、def[l]use: 1行目で自変数への代入に用いられている事を表し、use@pred(cond): 条件condで用いられている事を表す、とされる。

【0070】

最適化処理は例えば図40の変数表に基づいて行なわれる。最適化処理の一つ

は冗長ステートメントの削除である。

【0071】

冗長ステートメントの削除として、第1に、同一変数に対して、ステート遷移のカラム内でdefが2つ以上存在する場合には、1)又は2)の処理を行う。即ち、

1) 下記1-1), 1-2)をdefの後段に存在するpred(cond){...}の手前まで(pred(cond){...}の有無に関わらず)実施する。1-1):defの後段にuseを伴うdefがない場合は、最後のdefに対応するステートメントのみ残す。1-2):defの後段にuseを伴うdefがある場合は、useを伴うdefの後段にuseを伴わないdefがあれば、そのdefのみを残し、そうでなければuseを伴うdefの前段のdefとuseを伴うdefを残し、これを変化が無くなるまで繰り返し、残ったdefに対応するステートメントのみ残す。

2) defの後段にpred(cond){...}が無ければ終了し、あれば下記2-1), 2-2)を実施する。2-1):pred(cond){...}の条件がdefの結果を参照している場合には終了とする。2-2):pred(cond){...}の条件がdefの結果を参照していない場合は、1)へ分岐とする。

【0072】

冗長ステートメントの削除処理として、第2に、useがどのステート遷移にも存在しない変数は削除とする。

【0073】

上記処理手順により図40の変数表に対して冗長ステートメント削除を行ったとき、削除されるべきステートメントは図41に示される。同図において削除されるべきステートメントには斜め破線が明示されている。図42には冗長ステートメントが削除された結果の変数表が例示される。図43には冗長ステートメントが削除された結果をCFGで表している。

【0074】

最適化処理のもう一つはローカル変数の削除である。このローカル変数の削除処理として、第1に、各変数のステート遷移カラムに於いて、下記1)~3)を左から順次変化が無くなる迄実施する。即ち、

1) defの後段にpred(cond) {...}を挟まずuseが存在する場合には、1-1)、1-2)、1-3)、1-4)を行う。1-1): use自体がuse@predの場合は代入操作を実施し、defを削除し、1-2)の場合は@の変数がローカル変数でuse@predとして用いられている場合は代入操作を実施せず、1-3): @の変数がローカル変数でuse@predとして用いられていない場合は代入操作を実施し、defを削除し、1-4): @の変数が引数の場合は代入操作を実施し、defを削除する。

2) defの後段にpred(cond) {...}を挟んでuseが存在し、pred(cond) {...}の条件で用いられている変数が引数の場合は1-1)から1-4)を適用する。

3) defの後段にpred(cond) {...}を挟んでuseが存在し、pred(cond) {...}の条件で用いられている変数がローカル変数の場合には、3-1)、3-2)を行う。3-1): pred(cond) {...}の条件がdefの結果を参照していない場合は1-1)から1-4)を適用し、3-2): pred(cond) {...}の条件がdefの結果を参照している場合は代入操作を実施しない。

【0075】

ローカル変数の削除処理として、第2に、defがどのステート遷移カラムにも存在しない変数は削除し、代入操作後のCFGを再び解析して、変数表を更新する。

【0076】

図42の変数表に対してローカル変数削除を行ったとき、削除されるべき変数は図44に示される。同図において削除されるべき変数には斜め破線が明示されている。図45にはローカル変数削除処理が行なわれた結果をCFGで表している。

【0077】

図46には冗長ステートメント削除処理及びローカル変数削除処理が行なわれて最終的に更新された変数表が例示される。この変数表により、必要となるローカル変数が管理されることになる。図46において、defが存在しない部分では、出力変数とローカル変数の前置保持が必要である事が識別できる。ステートの遷移において当然前置保持されなければならないからである。従って、その部分

には、前置保持を必要とすることが容易に識別可能になる。図 4 7 に例示されるように、後工程の前置保持解析により、その部分に、前置保持 “retain” が追加されることになる。

【0078】

コードの最適化として更に、図 4 8 に例示されるような演算式の簡約化が行なわれる。

【0079】

《ステートマシン生成－前置保持解析》

ここからの説明は再度図 2 及び図 3 の仕様を満足させる回路設計の例に話を戻す。図 3 2 乃至図 4 8 では特別に簡素化した別の例を用いて前記コード最適化の処理を説明したが、図 3 1 に示されるステート割り当てが行われた後に、それと同様の最適化処理を施すことにより、図 4 9 の最適化後の C F G を得ることができ、また、図 5 0 の変数表が得られる。最適化処理後の図 5 0 の変数表には前置保持 “retain” は明示されていない。次に説明する前置保持解析で取得される。

【0080】

図 5 1 には前置保持解析のアルゴリズムが例示される。前置保持解析は、出力変数とローカル変数に対して、状態遷移のカラムにて def が全く存在しない場合、その状態遷移では前置保持が必要となる。また、def が存在したとしても、pred() が付加されている場合は、各出力変数・ローカル変数の各状態遷移に対して、図 5 1 に示されるような図を作成して、pred() による分岐の中でどの部分で前置保持が必要となるかを識別する。特に、レジスタ代入文識別処理で新たに追加したローカル変数に対しては、_i が付加されている変数のみに対して前置保持が必要かの解析を行う。また、例え、変数表に pred() の情報がなくても必要なら C F G を解析し直して付加する。

【0081】

図 5 1 に示される図の作成は、pred() の条件を分岐として、use、def 等のノードを持つ木を作成する事で行う。そして木の def より下位の部分木を削除し、最上位ノード以外で下位ノードに def が存在しないノードを前置保持が必要なノード

ドとして識別する。

【0082】

ここで、最上位ノードとは、木の根から一番距離に近いdefかuseのノードまでのノードとその兄弟ノード全てを指す。

【0083】

例えば、変数varの状態遷移 $ST_n \rightarrow ST_m$ での変数表からの情報が、`pred(cond_0) {pred(cond_1) {use@var_1[j], pred(cond_2) {def[k], pred(cond_3) {def[s]}}`}}である場合、図51のようになる。

【0084】

図52には前置保持解析の結果に対応する変数表が例示される。前置保持を要する部分には“retain”が追加される。

【0085】

変数表において“retain”の部分に追加すべき実際のコードは変数表から取得することができる。即ち、前置保持解析結果の変数表からの情報取得処理では、変数表のカラムでretainが挿入された、出力変数・ローカル変数を取得し、例えば変数名が

- 1) レジスタ代入文の左辺の変数の場合は、`sig = sig_0;`
 - 2) レジスタ代入文識別部で追加した変数であって、`_i` が付加されている変数の場合は、`sig_i = sig_0;`
 - 3) その他の変数の場合は、`nxt_sig = sig;`
- として記憶しておく。特に、retainにpred()が付加されている場合は、例えば、`pred(cond_0) {pred(cond_1) {pred(!cond_2) {retain}}}` に対しては、変数が3)の場合で変数名が sig の場合、`pred(cond_0) {pred(cond_1) {pred(!cond_2) {nxt_sig = sig}}}` として記憶する。以上の情報を変数表に上書き登録する。図53には“retain”が実際のコードで上書きされた変数表が例示される。更に、nxt_sigといった具合にnxt_を付加した変数を記憶しておく。図53の例の場合、nxt_を付加した変数は、a_tmpのみである。

【0086】

《ステートマシン生成—ステートマシン抽出》

次にステートマシン抽出処理 S5D について説明する。ステートマシン抽出処理 S5D では、割り当てた各ステートから深さ優先探索でクロック境界即ちステートであって初期ステートでないステートに到達するまで探索し、その探索で得られたループでも条件分岐でもラベル分岐でもないノードの情報を取得し、変数表の retain 情報とマージして、HDL 記述に用いるステートマシンの抽出を行う。例えば図 54 には開始ステート ST0 の例が示される。ステートの記述は、特に制限されないが、各ステートから DFS でコードを生成する。この場合ステート変数は、`nxt_state = ST0;`等の形式として、コード生成を行う。

【0087】

特に、retain 情報にて `nxt_` が付加された変数はもとの変数名ではなく `nxt_` が付加された変数名を用いて HDL 記述に用いるステートマシンの抽出を行う。また、信号と定数との & 演算はビット幅解析に用いたので、不要となるため削除する。尚、定数は入力左辺のビット数を勘案して HDL の 2 進表記に変換して HDL 記述に則した記述とする。

【0088】

変数表の retain 情報の取得では、各状態遷移カラムから、深さ優先探索を開始したステートと同じステートを開始ステートするカラムを全て取得し、retain が開始ステートのみに依存するか、到達ステートにも依存するか、または到達ステートと分岐条件に依存するかを識別し、開始ステートにのみ依存する場合以外は、retain 情報の `pred()` と CFG の分岐条件を比較する事で、HDL コードの適切な位置に retain 情報として変数表に格納した代入式を挿入する。図 55 には retain 情報に応ずるコードを変数表から抜き出してステートマシンの抽出に利用する様子が例示される。

【0089】

図 54 及び図 55 には開始ステート ST0 における HDL 記述に則したステートマシン記述の取得例が示される。図 56 及び図 57 の例は開始ステート ST1 における HDL 記述に則したステートマシン記述の取得例が示される。図 58 及び図 59 の例は開始ステート ST2 における HDL 記述に則したステートマシン記述の取得例が示される。

【0090】

《HDL記述生成処理》

HDL記述生成処理S6において、モジュール宣言は、回路動作記述部を含むC記述の関数宣言から、型とポインタを表す*を削除したものにclkとreset_nを加えたものをHDL記述として生成する。入出力宣言は、前記関数宣言での引数であって、代入式の左辺にのみ存在する変数を出力とし、代入式の右辺にのみ存在する変数を入力とし、ビット幅はC記述の記述内容で説明した方法で識別し、HDL記述として生成する。reg宣言はC記述に記載されていたローカル変数で、これまでの変換仮定で最終的に残った変数と、これまでの変換仮定で追加された変数とを識別し、clkとreset_nのreg宣言文とともにHDL記述として生成する。CFG生成過程で分岐条件に割り当てた変数のwire宣言のHDL記述を生成し、前記割り当てた変数への分岐条件の代入文をassign文としてHDL記述を生成する。また、割り当てたステートを2進数で表す為のparameter宣言文のHDL記述を生成する。

【0091】

また、レジスタ代入文に関しては、全てのレジスタ代入文とその右辺の変数宣言を取得し、例えば、取得した情報が

```
unsigned char sig1_latched = 0x00;
```

```
unsigned char sig2_latched = 0x00;
```

```
unsigned short out;
```

```
sig1_latched = $ sig1&0x03;
```

```
sig2_latched = $ sig2&0x13;
```

```
out          = $ exe_result&0x1FFF;
```

の場合、

```
always @ (posedge clk or negedge reset_n) begin
```

```
    if (!reset_n) begin
```

```
        sig1_latched_o <= 2' b00;
```

```
        sig2_latched_o <= 3' b000;
```

```
    end
```

```

else begin
    sig1_latched_o <= sig1_latched_i;
    sig2_latched_o <= sig2_latched_i;
    out_o          <= out_i;
end
end

```

のようなHDL記述を生成する。

【0092】

次いで、ステートマシン抽出部で得たnxt_が付加された変数の記憶を参照し、その変数の宣言部を取得し、例えば、この例の場合、a_tmpが対象となるが、

```
unsigned short nxt_a_tmp = 0x0000;
```

であり、reg宣言記述生成時に、

```
a_tmp = $ 0x7FFF&a;
```

なる代入から有効ビット幅が15ビットである事が解っているので、下記

```
always @(posedge clk or reset_n) begin
```

```
    if (!reset_n) begin
```

```
        state = ST0;
```

```
        a_tmp = 15' b0000000000000000;
```

```
    end
```

```
    else begin
```

```
        state = nxt_state;
```

```
        a_tmp = nxt_a_tmp;
```

```
    end
```

```
end
```

の記述を生成する。

【0093】

また、抽出されたステートマシンのHDL記述をつなげ、各ステートでの代入文の左辺に対して、レジスタ代入の左辺の変数とレジスタ代入文識別部で追加し

た変数には、対応する_oの変数を代入し、それ以外の変数には初期値を代入した文を作成し、nxt_state=ST0;なる文を作成し、case文のdefaultに対応する部分を作成し、それをつなげ、右辺の変数とwire宣言した変数をorで並べ、下記

```
always @ (state or c1 or c2 or valid_a_tmp_i or valid_a_tmp_o or valid_a_tmp or a_tmp or
```

```
valid_out_i or valid_out_o or out_i or out_o) begin
case(state[1:0])
endcase
end
```

の記述を生成し、case文の間につなげたHDL記述を挿入し、最後の行にendmoduleを付加する事でHDL記述を生成する。行数は付加しただけである。

【0094】

図60乃至図62にはHDL記述生成処理S6にて生成されたHDL記述が例示される。

【0095】

以上説明した設計方法によれば、以下の作用効果を得る。

【0096】

クロック境界を明示的にC記述内に挿入した擬似C記述を入力し、レジスタ代入文によるステートメントレベルでの並列記述を可能にした擬似C記述を入力するから、ストール動作を伴うパイプライン動作が表現可能である。

【0097】

ステートマシンを意識する事なくプログラム・レベルで機能設計を行う事ができるため、記述量が低減され、開発期間の短縮のみならず品質向上にも寄与する。

【0098】

また、一般のクロック境界を指定しないプログラム・レベルでの記述では表現できない、バス・インターフェース回路や調停回路の記述が可能となる。特に、レジスタ代入が記述可能である為、ステートメントレベルでの並列性を考慮した記述を行う事が可能であり、ストール動作を伴うパイプライン動作のような複雑

な回路動作をC記述よりも少ないコード量で容易に記述可能である。

【0099】

また、一般のCコンパイラでコンパイル可能なC記述へ変換する為、高速なシミュレーションが可能となり、機能検証工数の大幅な低減が可能となる。従って、機能設計における論理設計、論理検証の双方の大幅な工数削減が可能となる。

【0100】

高位合成ツールが不得意とする、サイクル精度を要求される例えば、キャッシュ・コントローラやDMAコントローラの開発に適用可能であり、設計期間の短縮に大きく寄与する。

【0101】

以上本発明者によってなされた発明を実施形態に基づいて具体的に説明したが、本発明はそれに限定されるものではなく、その要旨を逸脱しない範囲において種々変更可能であることは言うまでもない。

【0102】

例えば、以上説明したプログラム記述及び回路記述は一例であり種々の論理設計に適用することができる。HDLは必ずしもRTLに限定されない。プログラム記述言語はC言語に限定されず、その他の高級言語であってもよい。更にJava（登録商標）等の仮想マシン言語などを用いることも可能である。

【0103】

【発明の効果】

本願において開示される発明のうち代表的なものによって得られる効果を簡単に説明すれば下記の通りである。

【0104】

すなわち、本発明に係るコンパイラによれば、クロック境界を明示的に記述したプログラム記述からハードウェア記述を自動生成することができる。

【0105】

本発明に係るコンパイラによれば、ストール動作を伴うパイプライン動作が可能な回路のプログラム記述又は回路記述を容易に得る事ができる。

【0106】

本発明に係る論理回路の設計方法によれば、ストール動作を伴うパイプライン動作が可能な回路の設計を行うことができる。

【図面の簡単な説明】

【図 1】

本発明に係る論理回路の設計方法を例示するフローチャートである。

【図 2】

図 1 の設計方法を適用して設計すべき回路例を示すブロック図である。

【図 3】

図 2 の回路動作仕様を示すタイミングチャートである。

【図 4】

図 2 の設計対象回路の擬似 C プログラムを例示する説明図である。

【図 5】

レジスタ代入文識別処理（S 2）によって得られる追加変数宣言の記述とレジスタ代入文書き換えの記述を示す説明図である。

【図 6】

擬似 C 記述に基づく C F G 作成過程の一つの過程を示す説明図である。

【図 7】

擬似 C 記述に基づく C F G 作成過程の別の過程を示す説明図である。

【図 8】

擬似 C 記述に基づく C F G 作成過程の更に別の過程を示す説明図である。

【図 9】

擬似 C 記述に基づく C F G 作成過程の更に別の過程を示す説明図である。

【図 10】

擬似 C 記述に基づく C F G 作成過程の更に別の過程を示す説明図である。

【図 11】

擬似 C 記述に基づく C F G 作成過程の更に別の過程を示す説明図である。

【図 12】

擬似 C 記述に基づく C F G 作成過程の更に別の過程を示す説明図である。

【図 13】

擬似 C 記述に基づく C F G 作成過程の更に別の過程を示す説明図である。

【図 1 4】

擬似 C 記述に基づく C F G 作成過程の更に別の過程を示す説明図である。

【図 1 5】

擬似 C 記述に基づく C F G 作成過程の更に別の過程を示す説明図である。

【図 1 6】

擬似 C 記述に基づく C F G 作成過程の更に別の過程を示す説明図である。

【図 1 7】

擬似 C 記述に基づく C F G 作成過程の更に別の過程を示す説明図である。

【図 1 8】

擬似 C 記述に基づく C F G 作成過程の更に別の過程を示す説明図である。

【図 1 9】

擬似 C 記述に基づく C F G 作成過程の更に別の過程を示す説明図である。

【図 2 0】

擬似 C 記述に基づく C F G 作成過程の更に別の過程を示す説明図である。

【図 2 1】

擬似 C 記述に基づく C F G 作成過程の最終過程を示す説明図である。

【図 2 2】

説明を簡単化するために図 2 1 の C F G に対してクロック境界や分岐の始点・終点、及びループの始点・終点の情報を付加していない C F G を例示する説明図である。

【図 2 3】

図 2 2 の C F G に対するフラグ挿入状態を例示する説明図である。

【図 2 4】

レジスタ代入記述挿入文の挿入位置を C F G 上で例示する説明図である。

【図 2 5】

C 記述生成処理 (S 4) を経て得られる実行可能な変換 C 記述 (C プログラム) の最初の一部を例示する説明図である。

【図 2 6】

図 2 5 に続く実行可能な変換 C 記述 (C プログラム) の一部を例示する説明図である。

【図 2 7】

図 2 6 に続く実行可能な変換 C 記述 (C プログラム) の最後の部分を例示する説明図である。

【図 2 8】

ステート数削減処理の第 1 のルールを示す説明図である。

【図 2 9】

ステート数削減処理の第 2 のルールを示す説明図である。

【図 3 0】

図 2 2 の C F G に対してステート数削減を行った結果を例示する説明図である。

【図 3 1】

ステート数削減等の処理を行った C F G に対してステートの割り当てを行った状態を例示する説明図である。

【図 3 2】

前記コード最適化の処理を説明するために特別に簡素化した例としてコード最適化対象とされる擬似 C プログラムを示す説明図である。

【図 3 3】

図 3 2 の擬似 C プログラムに基づいて得られた C F G を例示する説明図である。

【図 3 4】

図 3 3 の C F G に対してステート割り当てが行なわれた状態を例示する説明図である。

【図 3 5】

図 3 4 の C F G に対してステートマシン生成のための変数表作成処理過程の最初の状態を例示する説明図である。

【図 3 6】

図 3 5 に続く変数表作成処理過程の次の状態を例示する説明図である。

【図 3 7】

図 3 6 に続く変数表作成処理過程の次の状態を例示する説明図である。

【図 3 8】

図 3 7 に続く変数表作成処理過程の次の状態を例示する説明図である。

【図 3 9】

図 3 8 に続く変数表作成処理過程の次の状態を例示する説明図である。

【図 4 0】

図 3 9 の生成過程を経て生成された変数表を例示する説明図である。

【図 4 1】

図 4 0 の変数表に対して冗長ステートメント削除を行ったとき、削除されるべきステートメントを例示する説明図である。

【図 4 2】

図 4 1 に対して冗長ステートメントが削除された結果の変数表を例示する説明図である。

【図 4 3】

冗長ステートメントが削除された結果を C F G で示す説明図である。

【図 4 4】

図 4 2 の変数表に対してローカル変数削除を行ったとき削除されるべき変数を例示する説明図である。

【図 4 5】

ローカル変数削除処理が行なわれた結果を C F G で示す説明図である。

【図 4 6】

冗長ステートメント削除処理及びローカル変数削除処理が行なわれて最終的に更新された変数表を例示する説明図である。

【図 4 7】

後工程の前置保持解析により変数表に前置保持 “retain “ の記述が追加された状態を例示する説明図である。

【図 4 8】

コードの最適化として更に演算式の簡約化を行った例を C F G で示す説明図で

ある。

【図 4 9】

図 3 2 乃至図 4 8 で特別に簡素化した別の例を用いて説明したコード最適化の処理を図 3 1 に示されるステート割り当てが行われた後に施すことによって得られる最適化後の C F G を例示する説明図である。

【図 5 0】

図 4 9 に対する最適化処理後の変数表を示す説明図である。

【図 5 1】

前置保持解析のアルゴリズムを例示する説明図である。

【図 5 2】

前置保持解析の結果に対応する変数表を示す説明図である。

【図 5 3】

図 5 2 に対し “retain” を実際のコードで上書きした変数表を示す説明図である。

【図 5 4】

開始ステート S T 0 におけるステートマシン抽出処理を示す説明図である。

【図 5 5】

図 5 4 に対し retain 情報に応ずるコードを変数表から抜き出してステートマシンの抽出に利用する様子を示す説明図である。

【図 5 6】

開始ステート S T 1 におけるステートマシン抽出処理を示す説明図である。

【図 5 7】

図 5 6 に対し retain 情報に応ずるコードを変数表から抜き出してステートマシンの抽出に利用する様子を示す説明図である。

【図 5 8】

開始ステート S T 2 におけるステートマシン抽出処理を示す説明図である。

【図 5 9】

図 5 8 に対し retain 情報に応ずるコードを変数表から抜き出してステートマシンの抽出に利用する様子を示す説明図である。

【図 6 0】

HDL 記述生成処理（S 6）にて生成された HDL 記述の最初の一部を示す説明図である。

【図 6 1】

図 6 0 に続く HDL 記述の一部を示す説明図である。

【図 6 2】

図 6 1 に続く HDL 記述の最後の部分を示す説明図である。

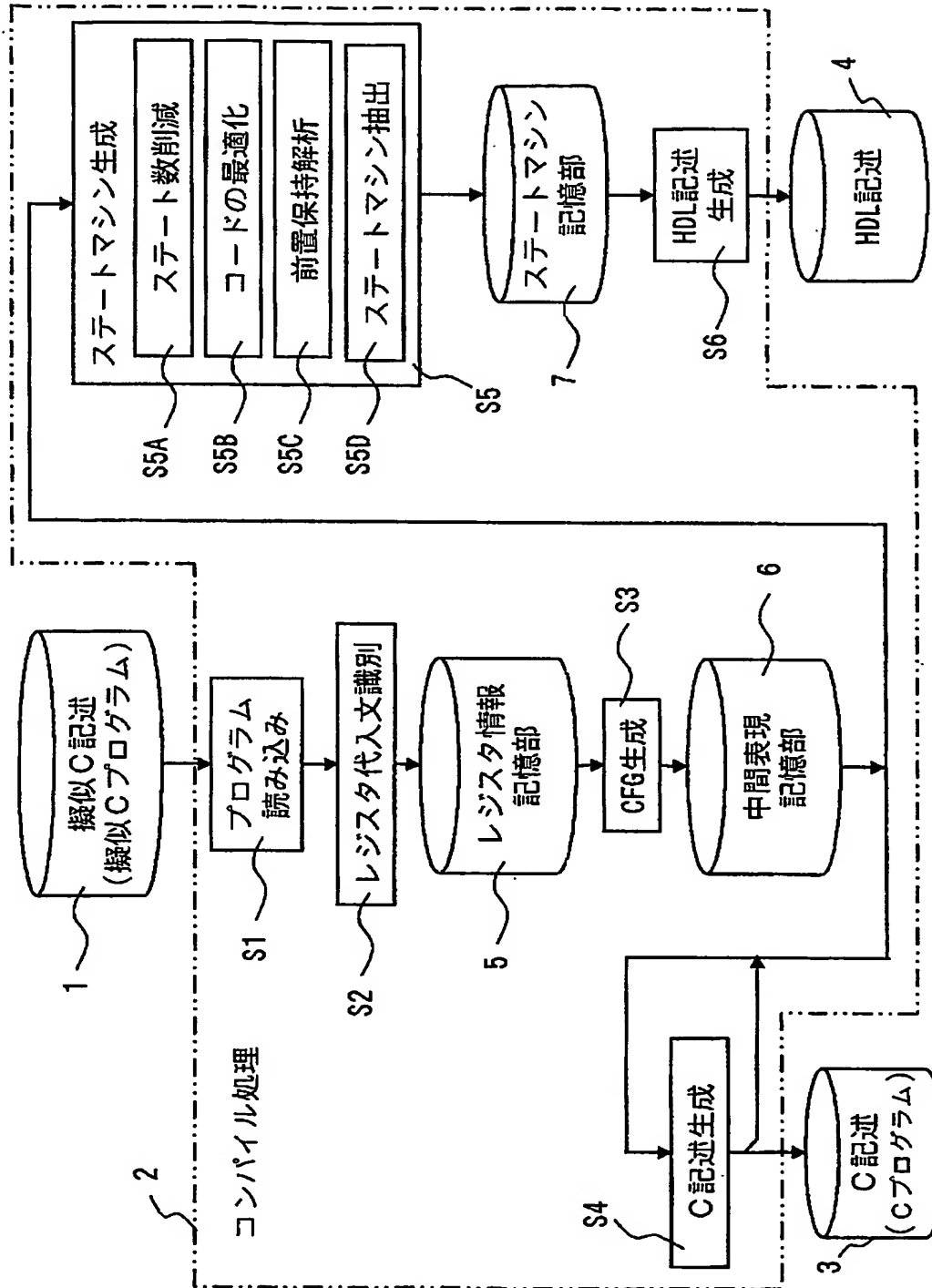
【符号の説明】

- 1 擬似 C 記述（擬似 C プログラム）
- 2 コンパイル処理
- 3 C 記述（C プログラム）
- 4 HDL 記述
- 5 レジスタ情報記憶部
- 6 中間表現記憶部
- 7 ステートマシン記憶部

【書類名】 図面

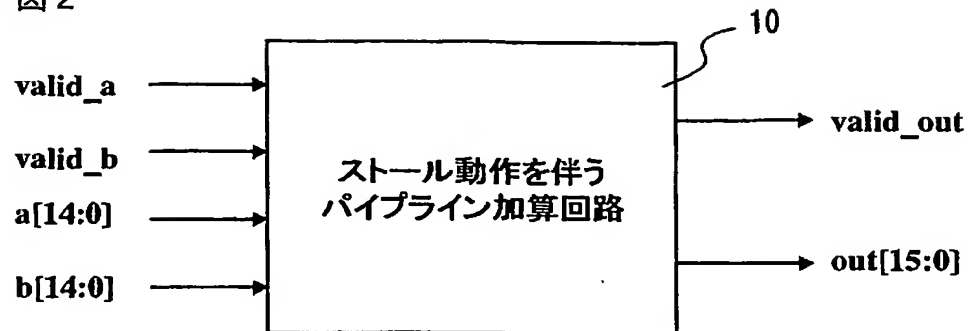
【図1】

図1

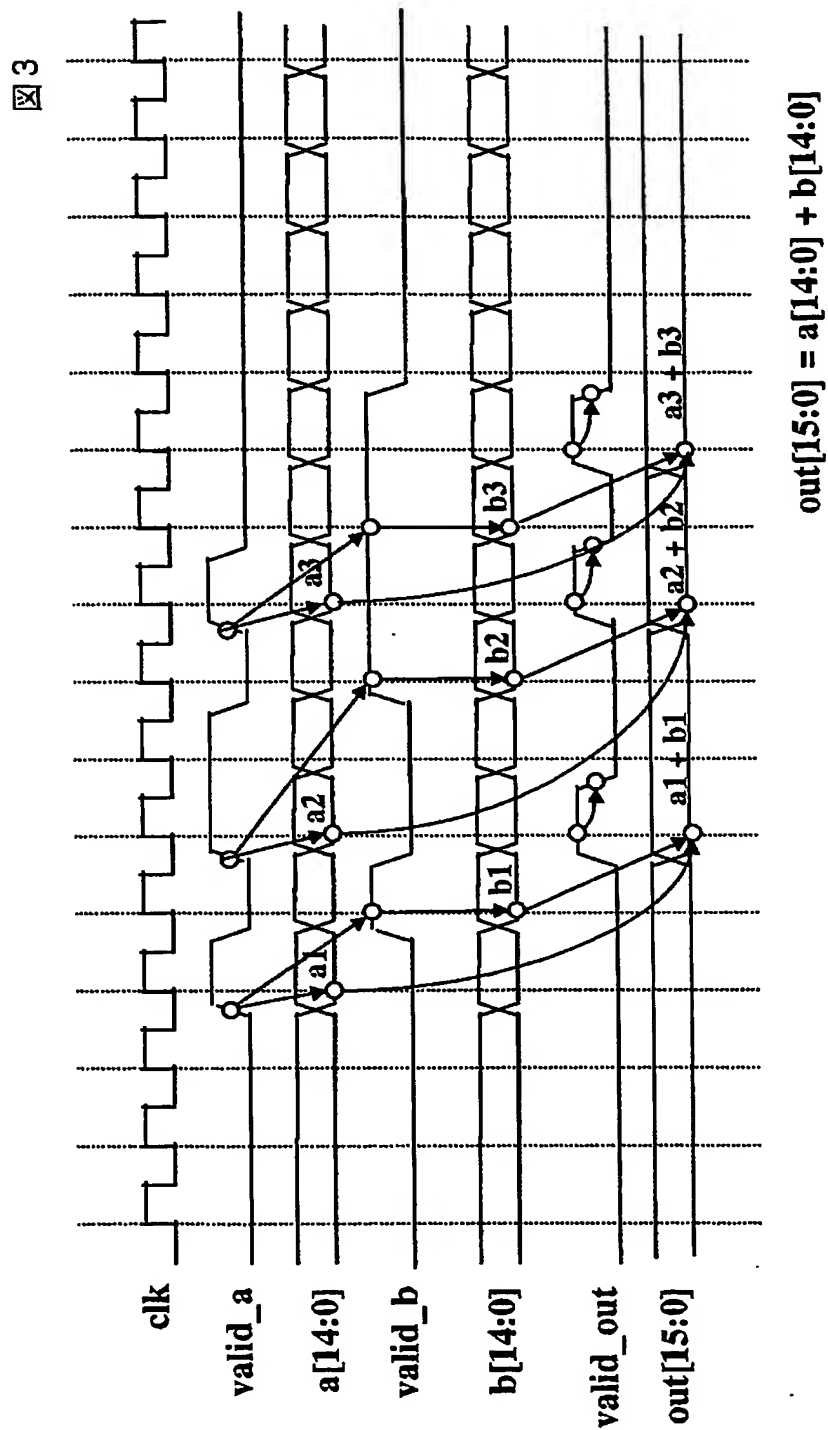


【図 2】

図 2



【図 3】



【図 4】

図 4

```

1  #include <stdio.h>
2  void pipeline(unsigned short valid_a, unsigned short valid_b,
3               unsigned short valid_b,
4               unsigned short a,
5               unsigned short b,
6               unsigned short *out,
7               unsigned short *valid_out);
8  main() {
9      unsigned short valid_a, valid_b,
10      a, b, *out, *valid_out;
11      *out = 0x0000;
12      *valid_out = 0x0000;
13      pipeline(valid_a, valid_b, a, b, out, valid_out);
14  }

15 void pipeline(unsigned short valid_a, unsigned short valid_b,
16               unsigned short a, unsigned short b,
17               unsigned short *out, unsigned short *valid_out) {
18     unsigned short valid_a_tmp = 0x0000;
19     unsigned short a_tmp = 0x0000;
20     unsigned short b_tmp = 0x0000;
21     while (1) {
22         valid_a_tmp = $ 0x0001 & valid_a;
23         if ((0x0001 & valid_a_tmp == 0x0000) && (0x0001 & valid_a == 0x0001)) {
24             a_tmp = 0x7FFF & a;
25             $
26         L:
27             if (0x0001 & valid_b == 0x0001) b_tmp = 0x7FFF & b;
28             else $ goto L;
29             *out = $ (a_tmp + b_tmp);
30             *valid_out = $ 0x0001;
31         } else {
32             $
33             *valid_out = $ 0x0000;
34         }
35     }
36 }

```

11(回路動作記述部)

【図 5】

図 5

```

1  #include <stdio.h>
2  void pipeline(unsigned short valid_a, unsigned short valid_b,
3               unsigned short valid_b,
4               unsigned short a,
5               unsigned short b,
6               unsigned short *out,
7               unsigned short *valid_out);
8  main() {
9      unsigned short valid_a, valid_b,
10         a, b, *out, *valid_out;
11      *out = 0x0000;
12      *valid_out = 0x0000;
13      pipeline(valid_a, valid_b, a, b, out, valid_out);
14  }

```

追加変数宣言

```

15 void pipeline(unsigned short valid_a, unsigned short valid_b,
16               unsigned short a, unsigned short b,
17               unsigned short *out, unsigned short *valid_out) {
18     unsigned short valid_a_tmp = 0x0000;
19     unsigned short a_tmp = 0x0000;
20     unsigned short b_tmp = 0x0000;
21     unsigned short valid_a_tmp_i;
22     unsigned short valid_a_tmp_o = 0x0000;
23     unsigned short out_i;
24     unsigned short out_o = 0x0000;
25     unsigned short valid_out_i;
26     unsigned short valid_out_o = 0x0000;
27     while (1) {
28         valid_a_tmp_j = 0x0001 & valid_a;
29         valid_a_tmp = valid_a_tmp_o;
30         if ((0x0001 & valid_a_tmp == 0x0000) && (0x0001 & valid_a == 0x0001)) {
31             a_tmp = 0x7FFF & a;
32             $
33             L:
34             if (0x0001 & valid_b == 0x0001) b_tmp = 0x7FFF & b;
35             else $ goto L;
36             out_j = (a_tmp + b_tmp);
37             *out = out_o;
38             valid_out_j = 0x0001;
39             *valid_out = valid_out_o;
40         } else {
41             $
42             valid_out_j = 0x0001;
43             *valid_out = valid_out_o;
44         }
45     }
46 }
47 }

```

13(レジスタ代入文書き換え文)

【図 6】

図 6

27 while



ループ文スタック

27 while

分岐文スタック

null

【図 7】

図 7

27 while



```
28 valid_a_tmp_i = 0x0001&valid_a;
29 valid_a_tmp_o = valid_a_tmp_o;
```

ループ文スタック

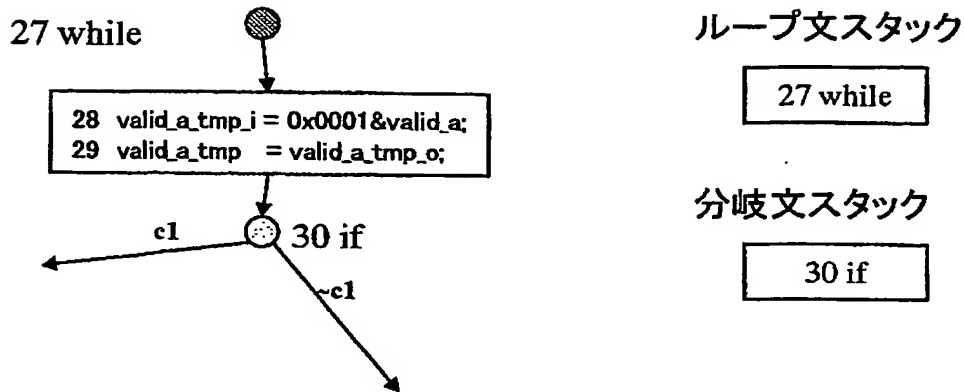
27 while

分岐文スタック

null

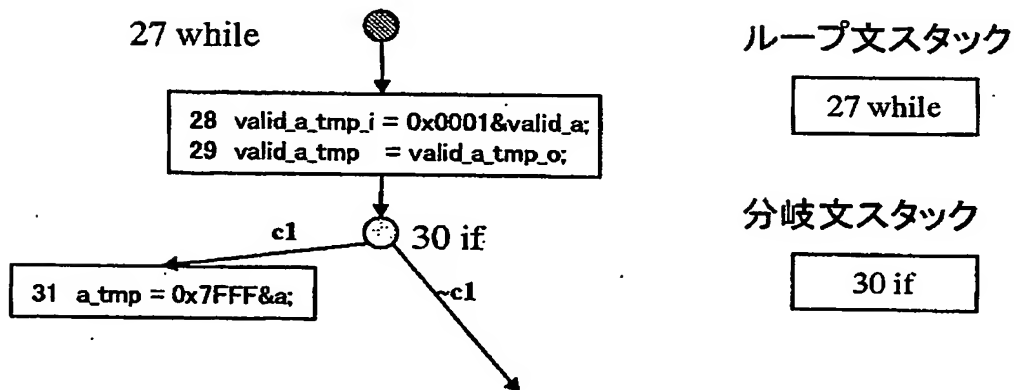
【図 8】

図 8


$$\left[C1 = (0x0001 \& \text{valid_a_tmp} == 0x0000) \&\& (0x0001 \& \text{valid_a} == 0x0001) \right]$$

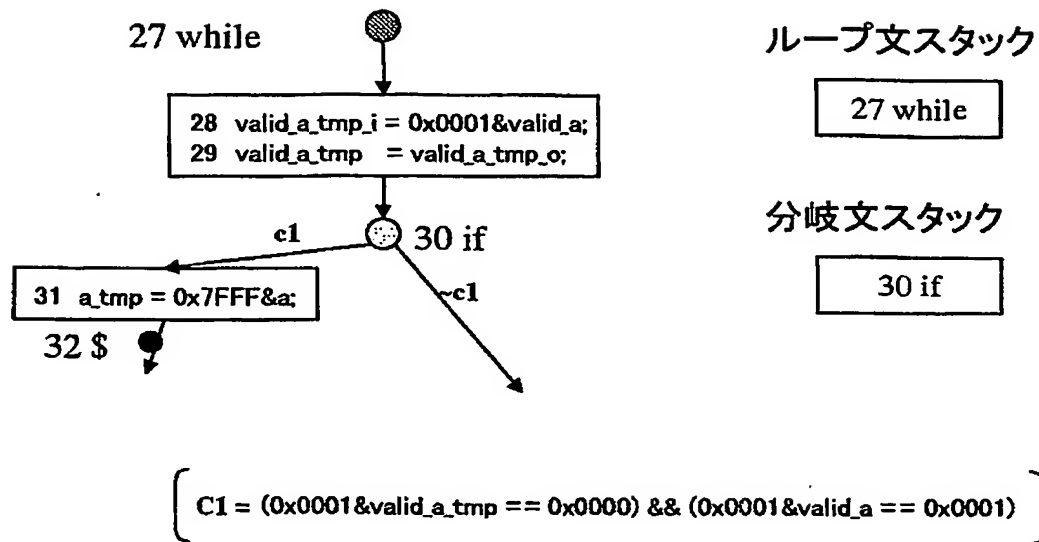
【図 9】

図 9


$$\left[C1 = (0x0001 \& \text{valid_a_tmp} == 0x0000) \&\& (0x0001 \& \text{valid_a} == 0x0001) \right]$$

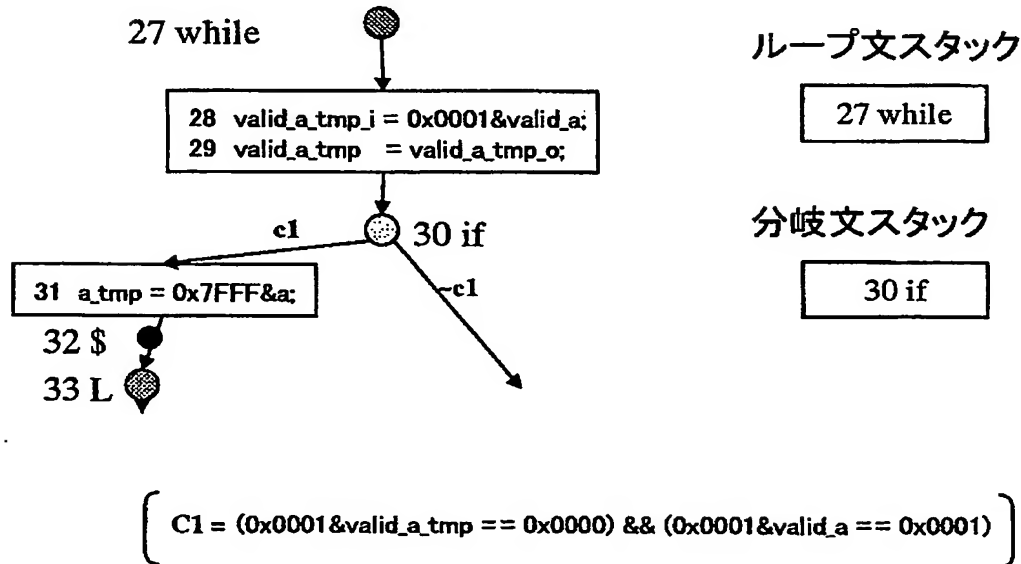
【図 10】

図 10



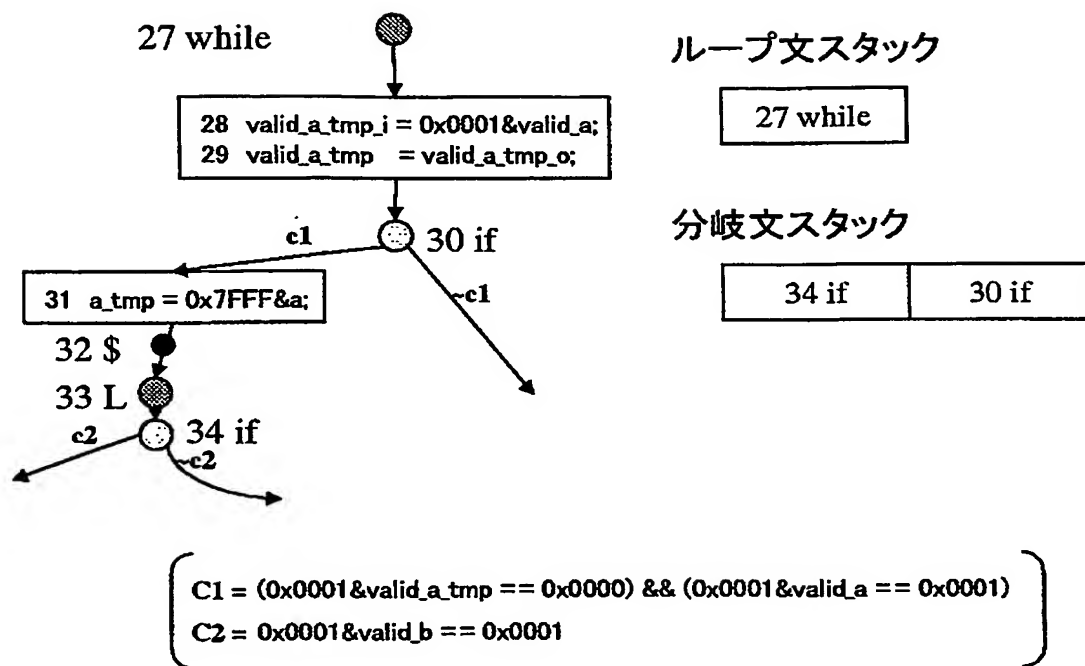
【図 11】

図 11

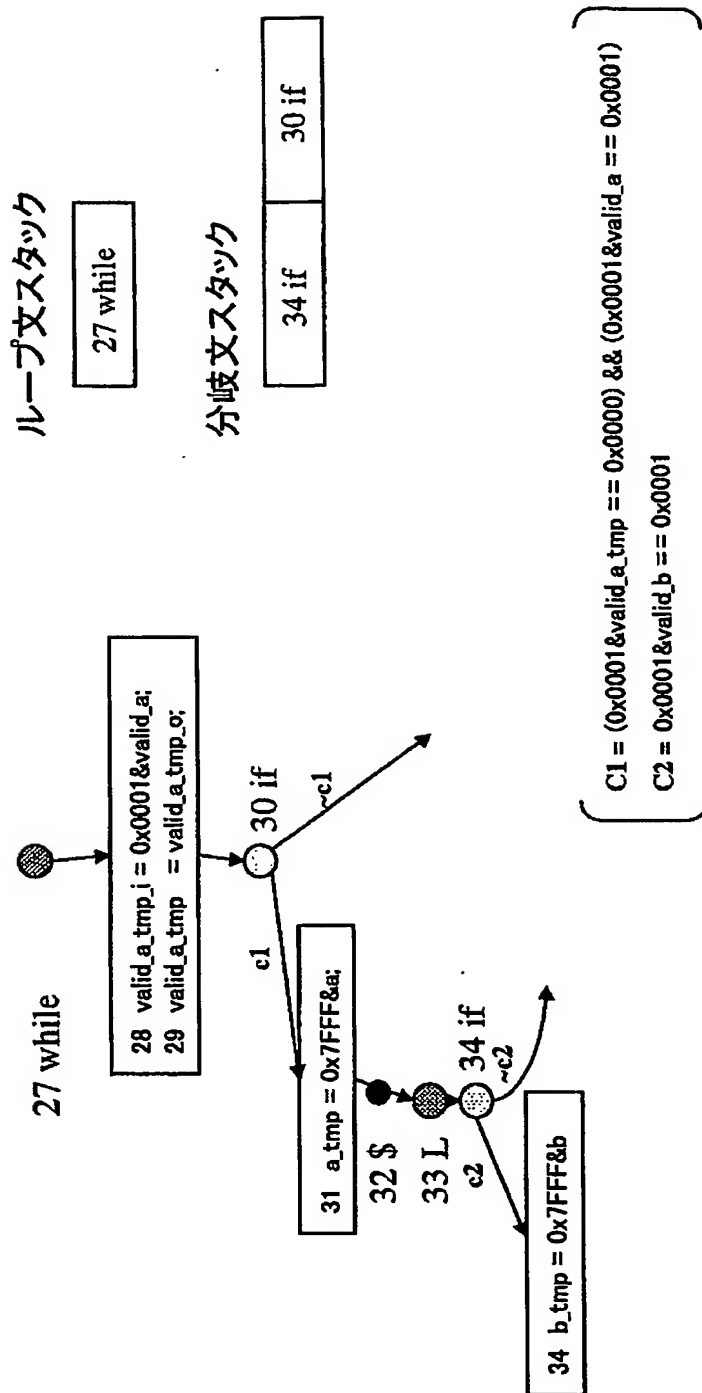



【図 12】

図 12



【図 13】



【図 15】

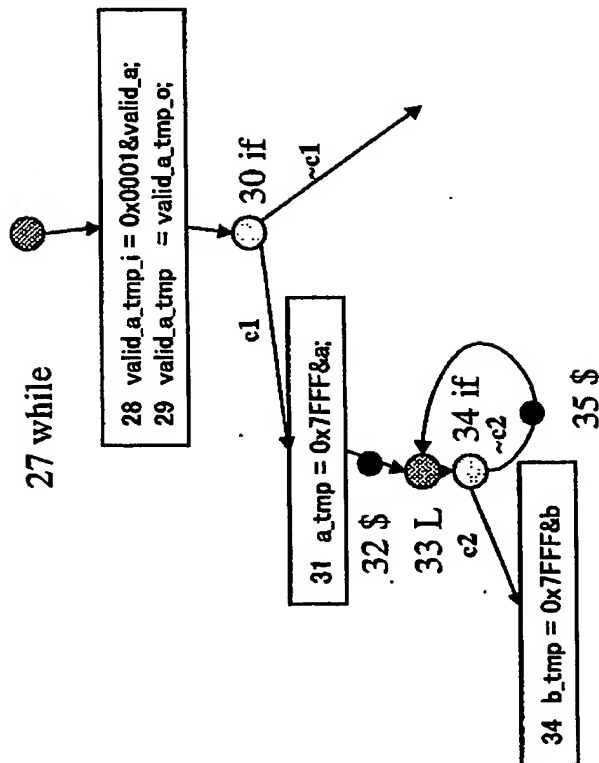
図 15

ループ文スタック

27 while

分岐文スタック

34 if 30 if



$$\left[\begin{array}{l} C1 = (0x0001 \& \text{valid_a_tmp} == 0x0000) \& \& (0x0001 \& \text{valid_a} == 0x0001) \\ C2 = 0x0001 \& \text{valid_b} == 0x0001 \end{array} \right]$$

【図 16】

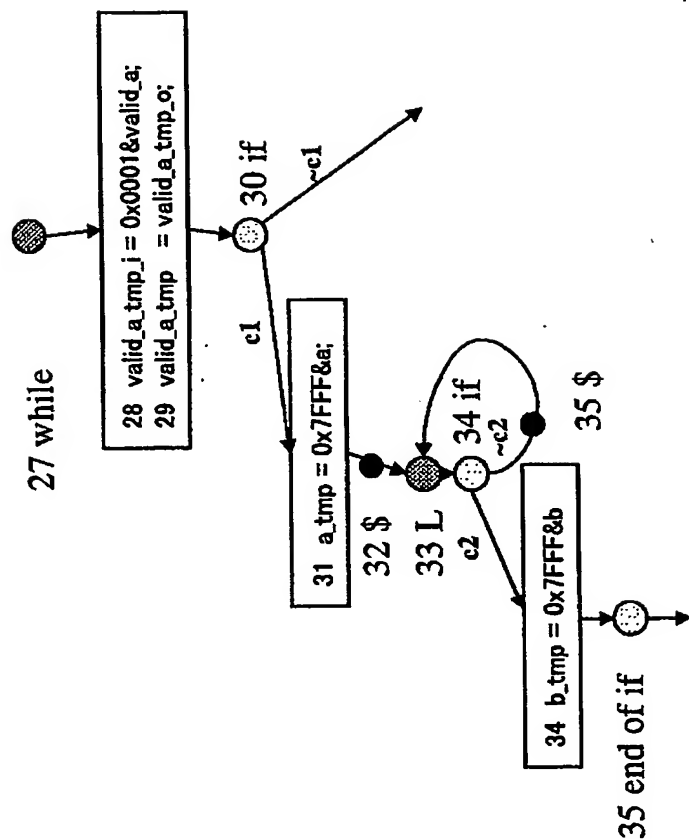
図 16

ループスタック

27 while

分岐スタック

30 if



$$C1 = (0x0001 \& \text{valid_a_tmp} == 0x0000) \&\& (0x0001 \& \text{valid_a} == 0x0001)$$

$$C2 = 0x0001 \& \text{valid_b} == 0x0001$$

【図 17】

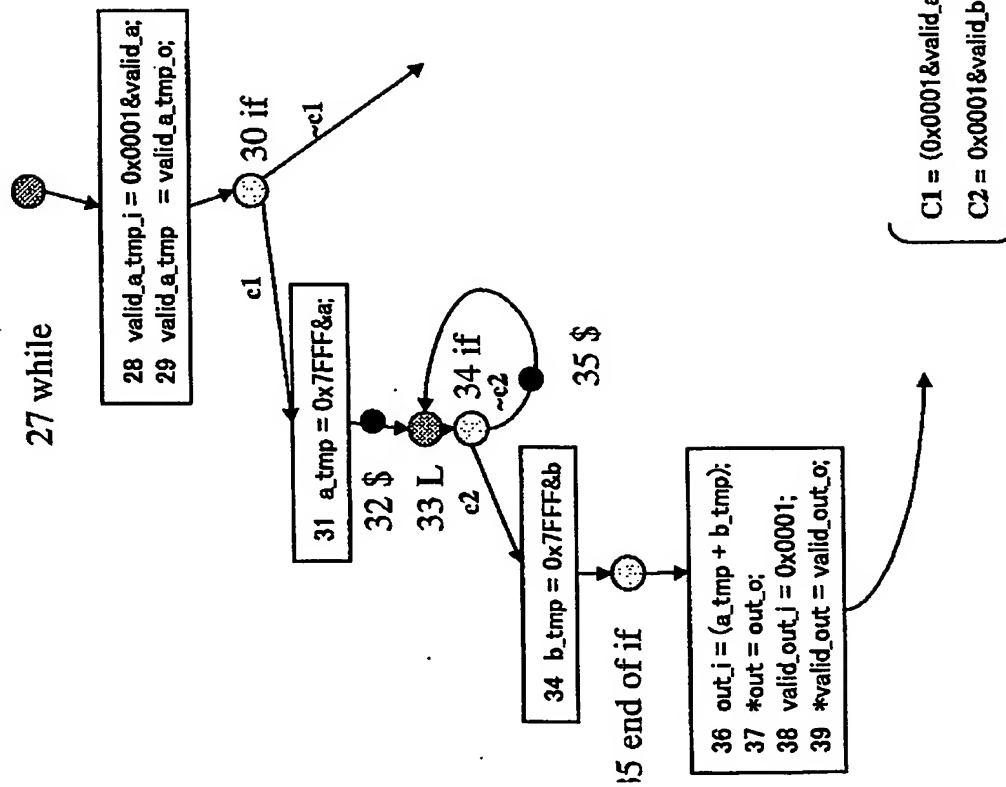
図 17

ループ文スタック

27 while

分岐文スタック

30 if



【図 19】

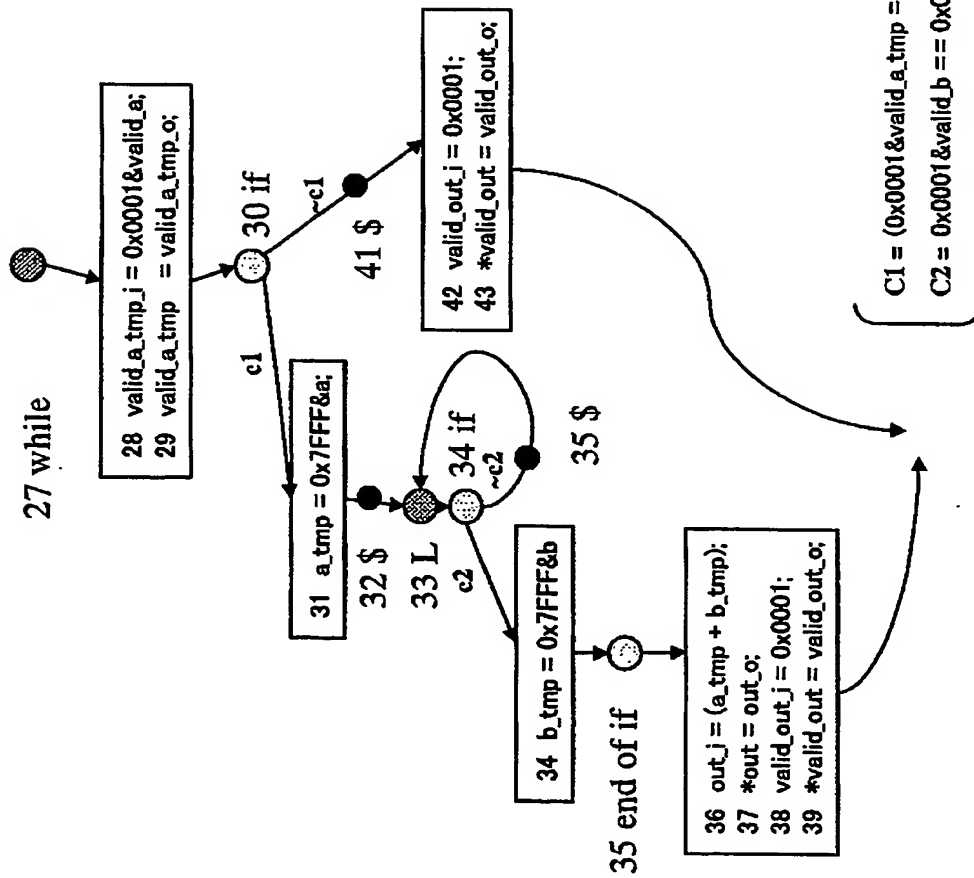
図 19

ループ文スタック

27 while

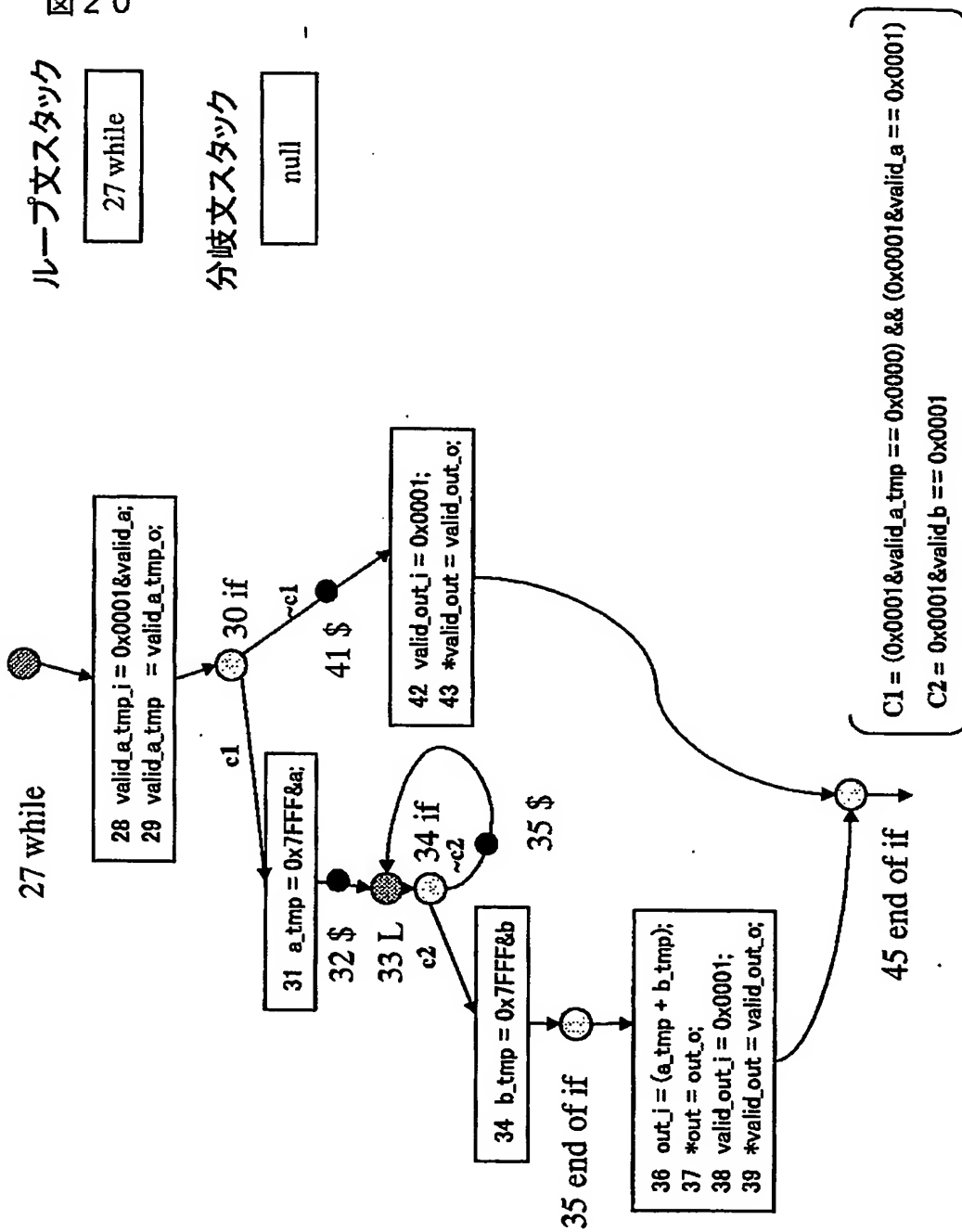
分岐文スタック

30 if

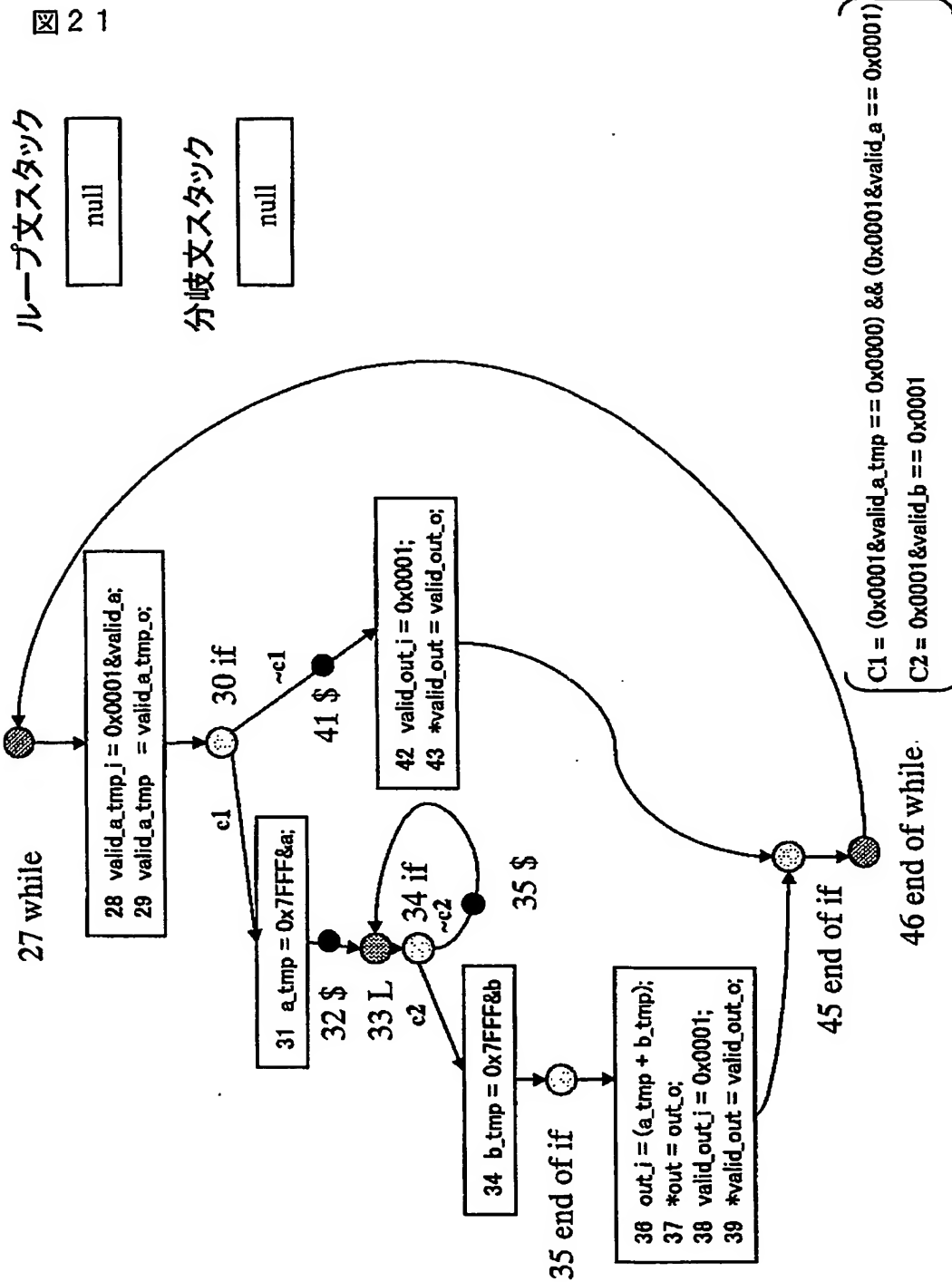


【図 20】

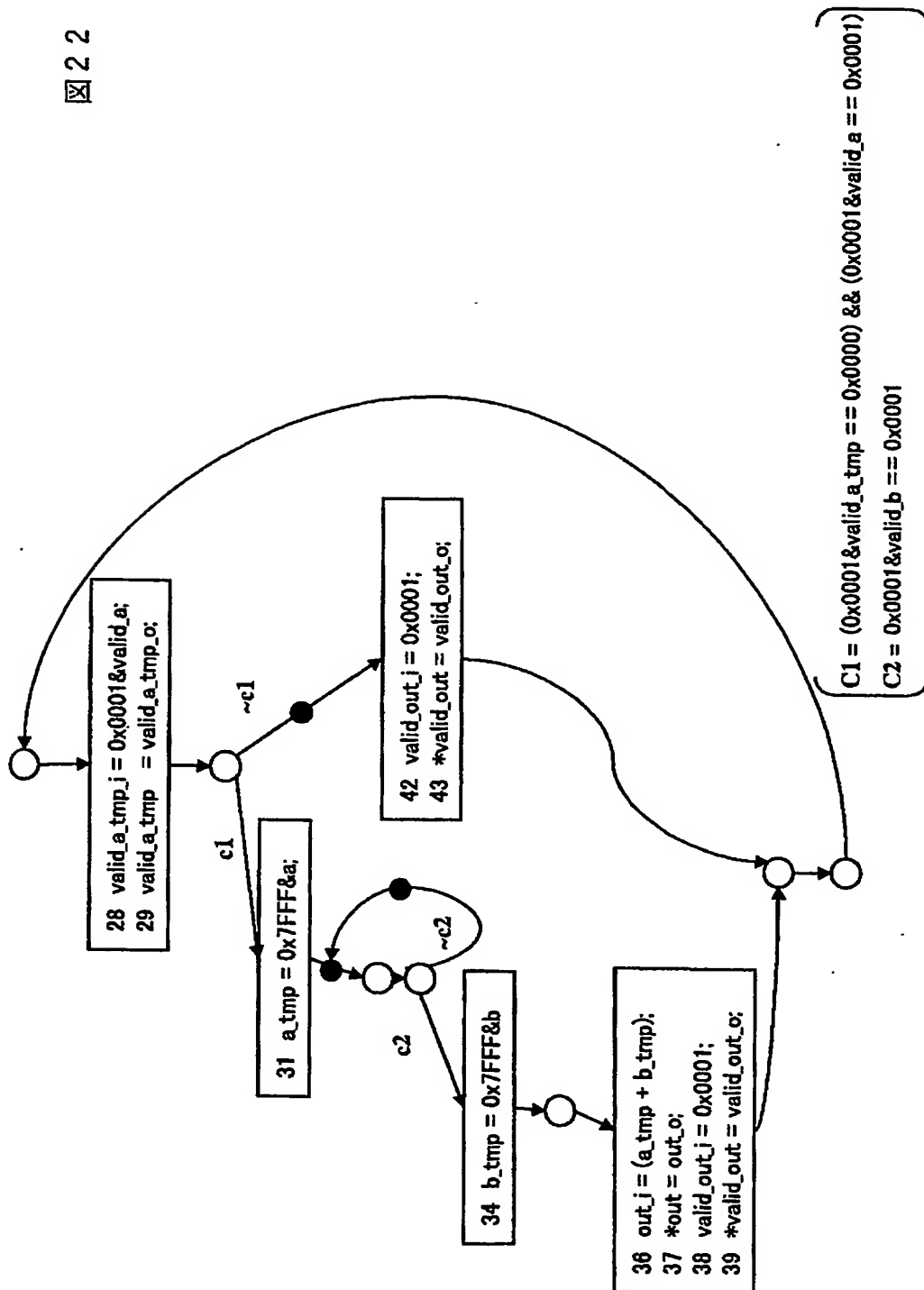
図 20



【図 21】

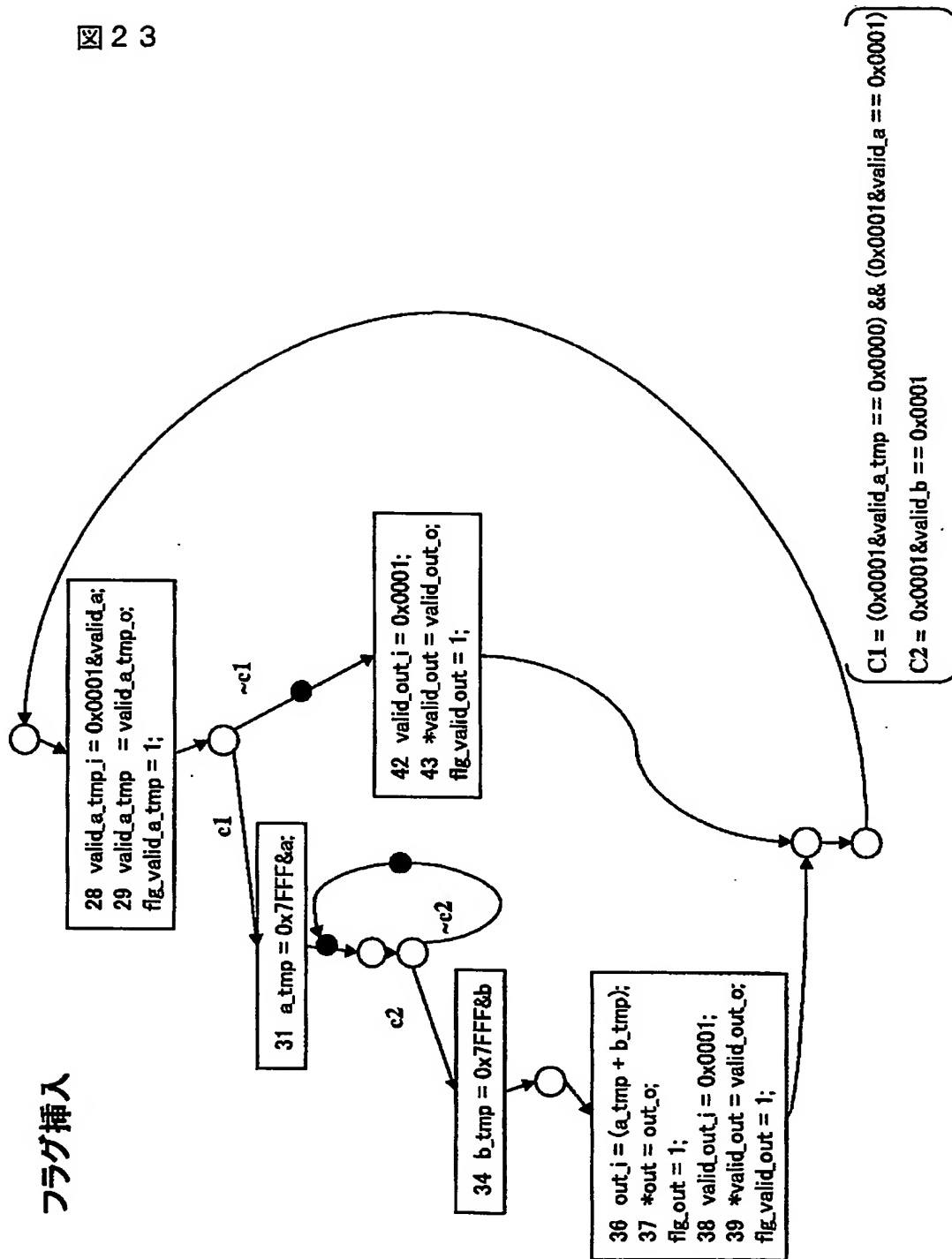


【図 22】



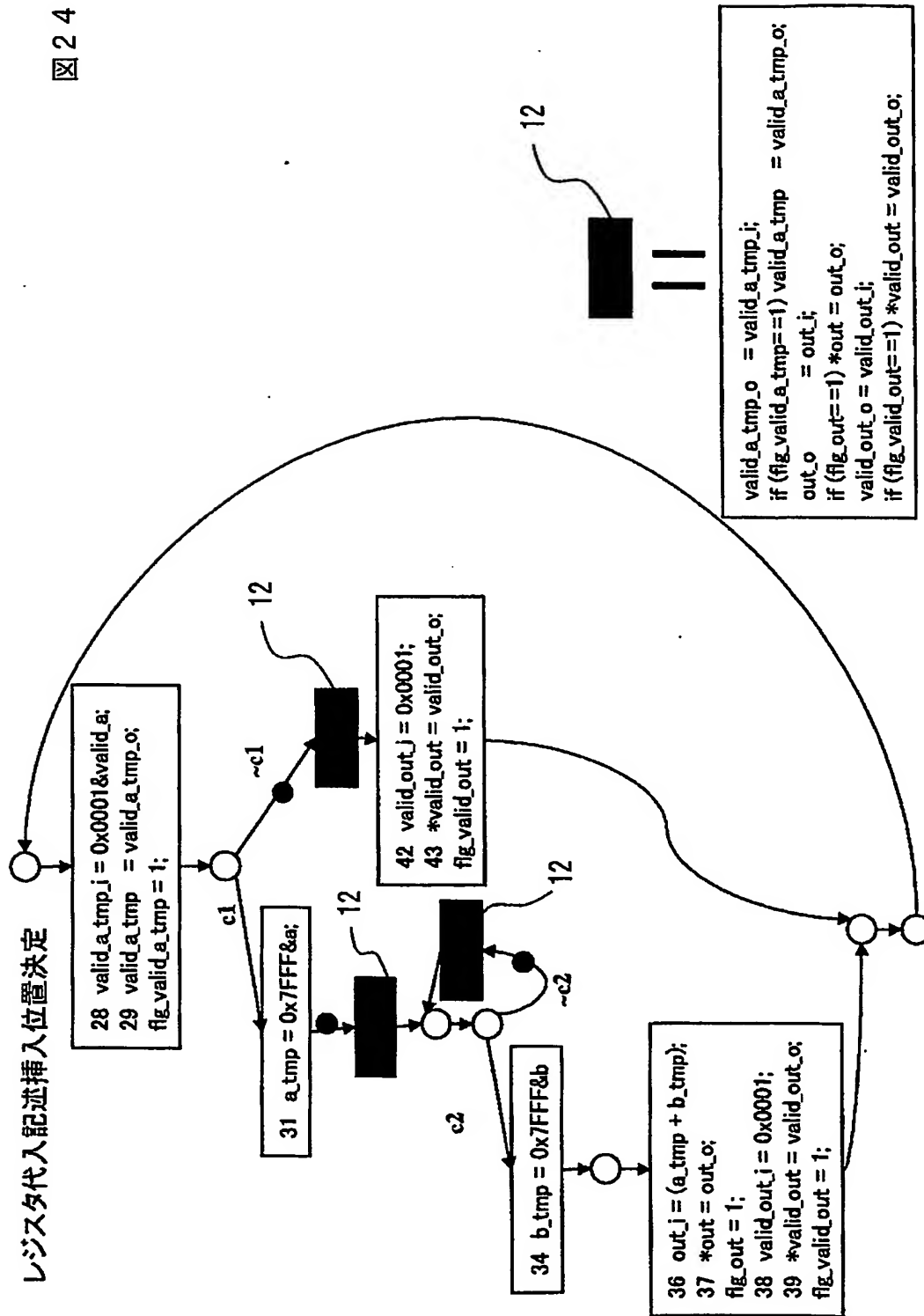
【図 23】

図 23



【図 24】

図 24



【図 25】

図 25

```

1  #include <stdio.h>
2  void pipeline(unsigned short valid_a,
3               unsigned short valid_b,
4               unsigned short a,
5               unsigned short b,
6               unsigned short *out,
7               unsigned short *valid_out);
8  main() {
9      unsigned short valid_a, valid_b,
10         a, b, *out, *valid_out;
11      *out = 0x0000;
12      *valid_out = 0x0000;
13      pipeline(valid_a, valid_b, a, b, out, valid_out);
14  }

15 void pipeline(unsigned short valid_a, unsigned short valid_b,
16               unsigned short a, unsigned short b,
17               unsigned short *out, unsigned short *valid_out) {
18     unsigned short valid_a_tmp = 0x0000;
19     unsigned short a_tmp = 0x0000;
20     unsigned short b_tmp = 0x0000;
21     /* Added variables */
22     unsigned short valid_a_tmp_i;
23     unsigned short valid_a_tmp_o = 0x0000;
24     unsigned short valid_out_i;
25     unsigned short valid_out_o = 0x0000;
26     unsigned short out_i;
27     unsigned short out_o = 0x0000;
28     unsigned short flg_valid_a_tmp = 0x0000;
29     unsigned short flg_valid_out = 0x0000;

```

【図 2 6】

図 2 6

```
30 while (1) {  
    /* valid_a_tmp = $ valid_a; */  
31 valid_a_tmp_i = 0x0001&valid_a;    /* Refined */  
32 valid_a_tmp_o = valid_a_tmp_i;    /* Refined */  
33 flg_valid_a_tmp = 1;  
34 if ((0x0001&valid_a_tmp == 0x0000) && (0x0001&valid_a == 0x0001)) {  
35     a_tmp = 0x7FFF&a;  
    /* $ */  
    /* BEGIN : Register Assignment */  
36     valid_a_tmp_o = valid_a_tmp_i;  
37     if (flg_value_a_tmp == 1) valid_a_tmp_o = valid_a_tmp_o;  
38     out_o = out_i;  
39     if (flg_out==1) *out = out_o;  
40     valid_out_o = valid_out_i;  
41     if (flg_valid_out==1) *valid_out = valid_out_o;  
    /* END : Register Assignment */  
42 L :
```

【图 27】

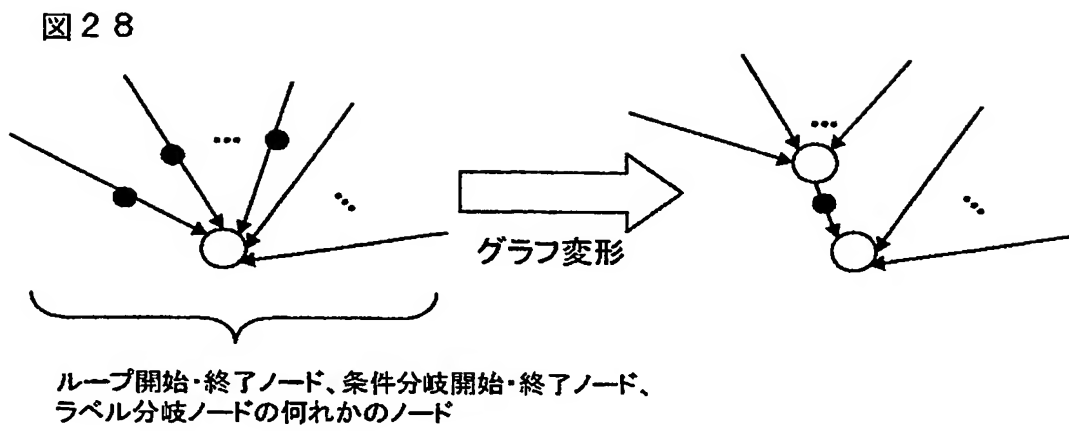
图 27

```

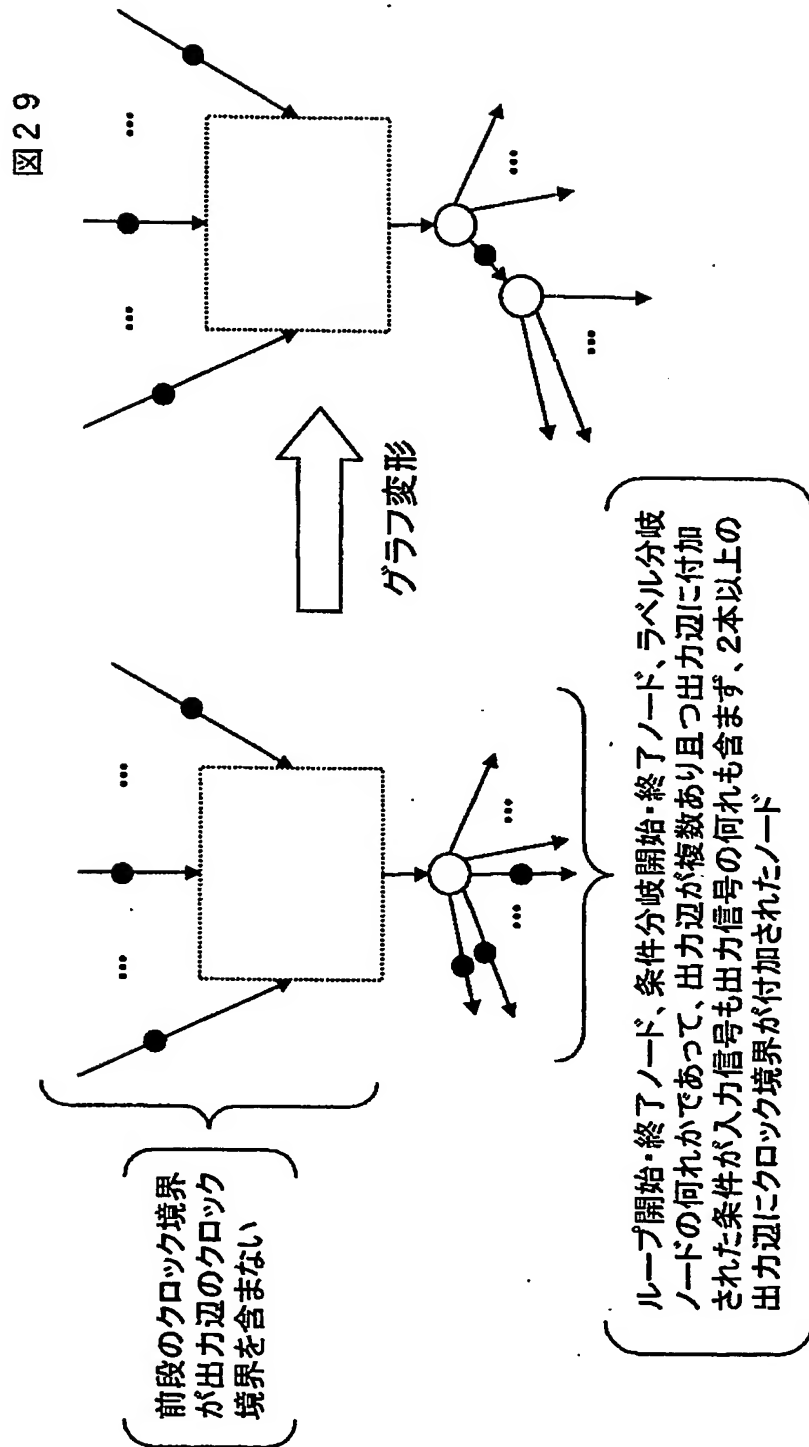
43 if (0x0001&valid_b == 0x0001) b_tmp = 0x7FFF&b;
44 else {
45     /* $ */
46     /* BEGIN : Register Assignment */
47     valid_a_tmp_o = valid_a_tmp_i;
48     valid_a_tmp = valid_a_tmp_o;
49     out_o = out_i;
50     if (flag_out==1) *out = out_o;
51     valid_out_o = valid_out_i;
52     if (flag_valid_out==1) *valid_out = valid_out_o;
53     /* END : Register Assignment */
54     /* *valid_out = $ 0x0000; */
55     goto L;
56     /* *out = $ (a_tmp + b_tmp); */
57     out_j = a_tmp + b_tmp;
58     /*out = out_o; */
59     flag_out = 1;
60     /* *valid_out = $ 0x0001; */
61     valid_out_j = 0x0001;
62     *valid_out = valid_out_o;
63     flag_valid_out = 1;
64 }
65
66
67
68
69
70
71

```

【図 28】

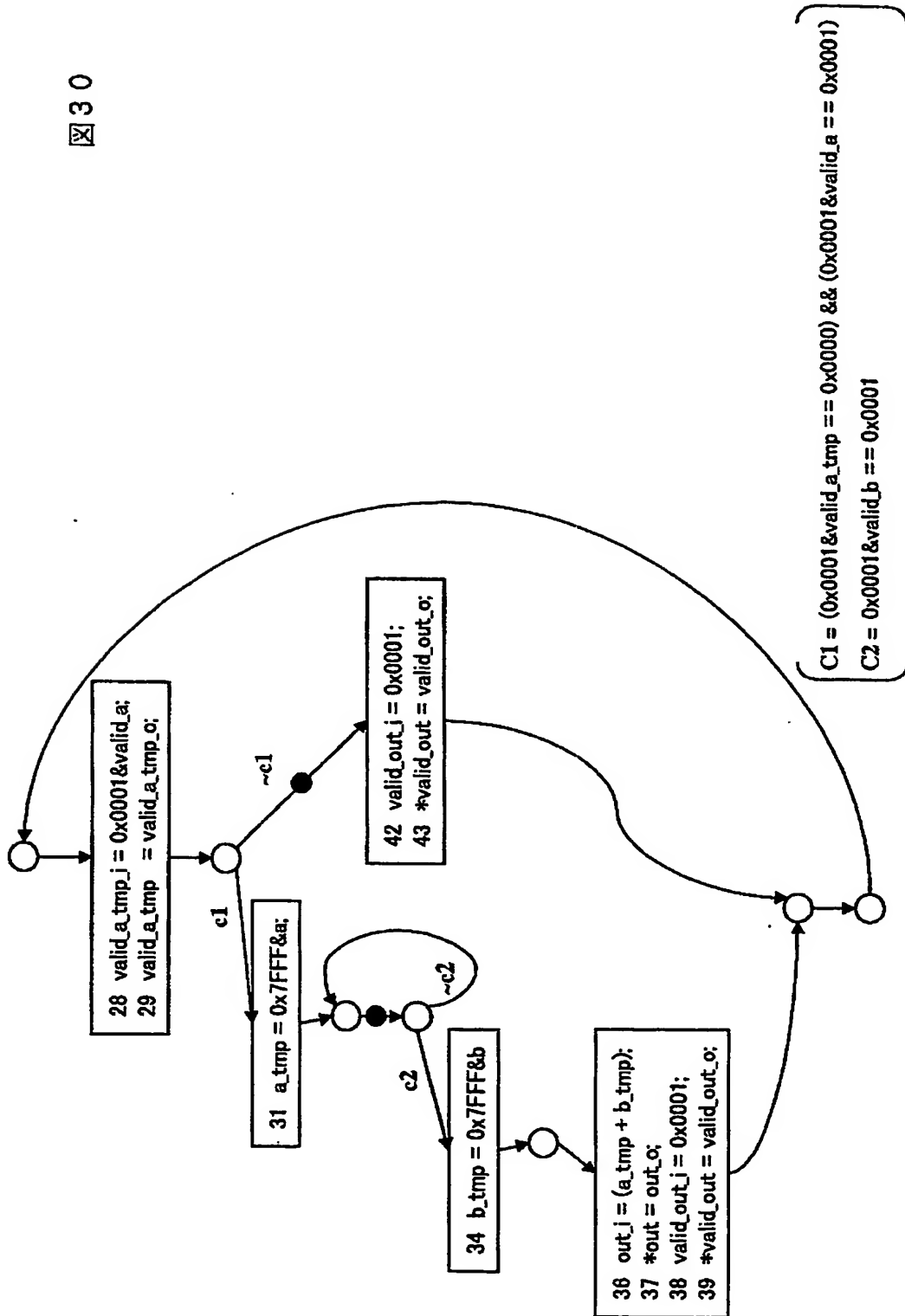


【図 29】



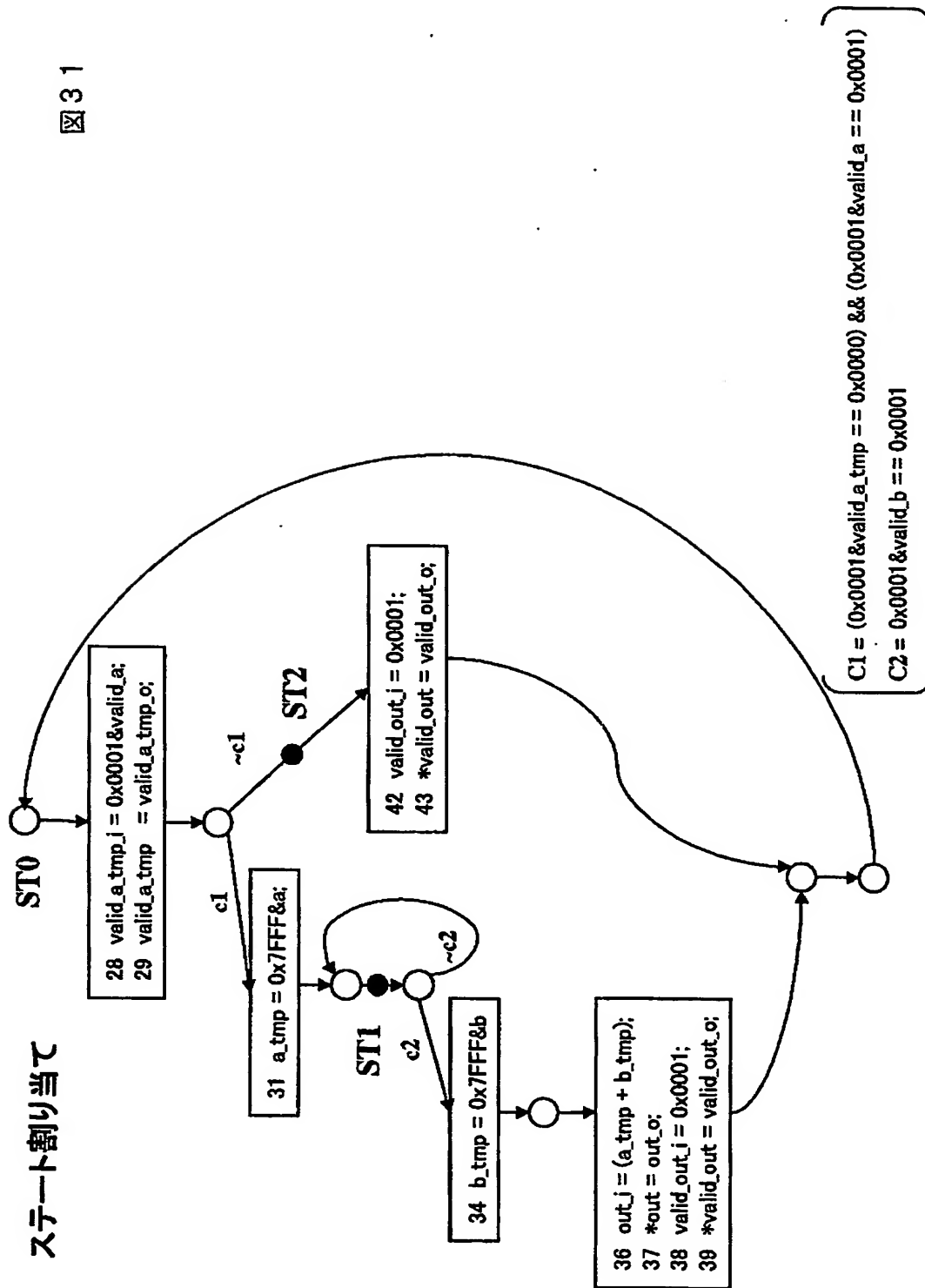
【図 30】

図 30



【図 31】

図 31



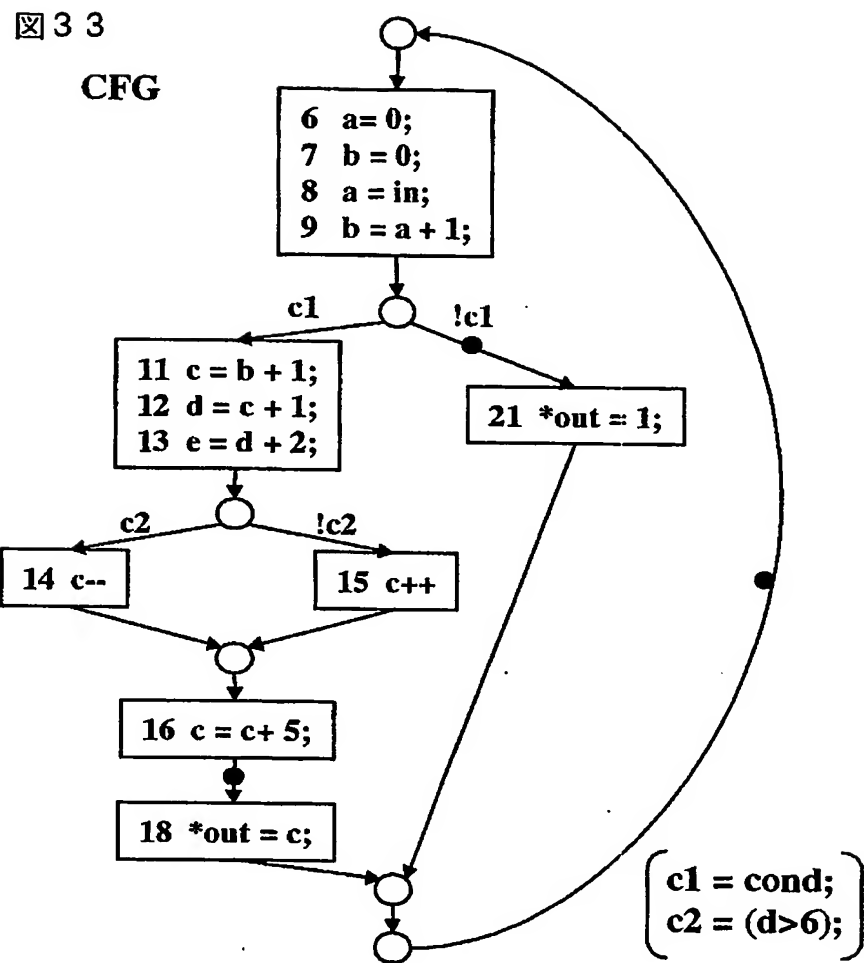
【図 3 2】

図 3 2

コード最適化 (別の例)

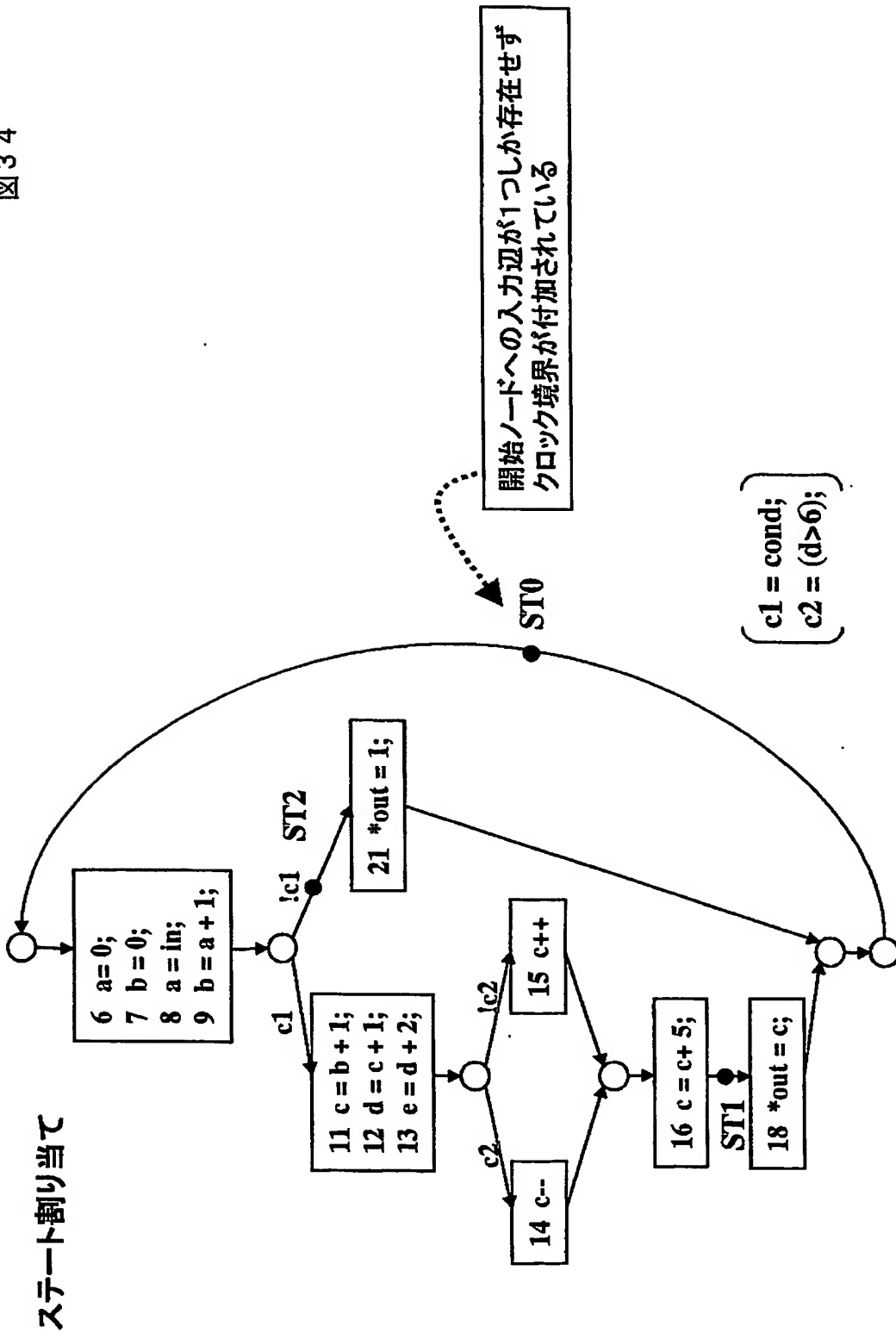
```
1 void foo(unsigned short in,  
2         unsigned short cond,  
3         unsigned short *out) {  
4     unsigned short a, b, c, d, e;  
5     while(1) {  
6         a = 0;  
7         b = 0;  
8         a = in;  
9         b = a + 1;  
10        if (cond) {  
11            c = b + 1;  
12            d = c + 1;  
13            e = d + 2;  
14            if (d > 6) c--;  
15            else c++;  
16            c = c + 5;  
17            $  
18            *out = c;  
19        } else {  
20            $  
21            *out = 1;  
22        }  
23        $  
24    }
```


【図 3 3】



【図 34】

図 34



【図 35】

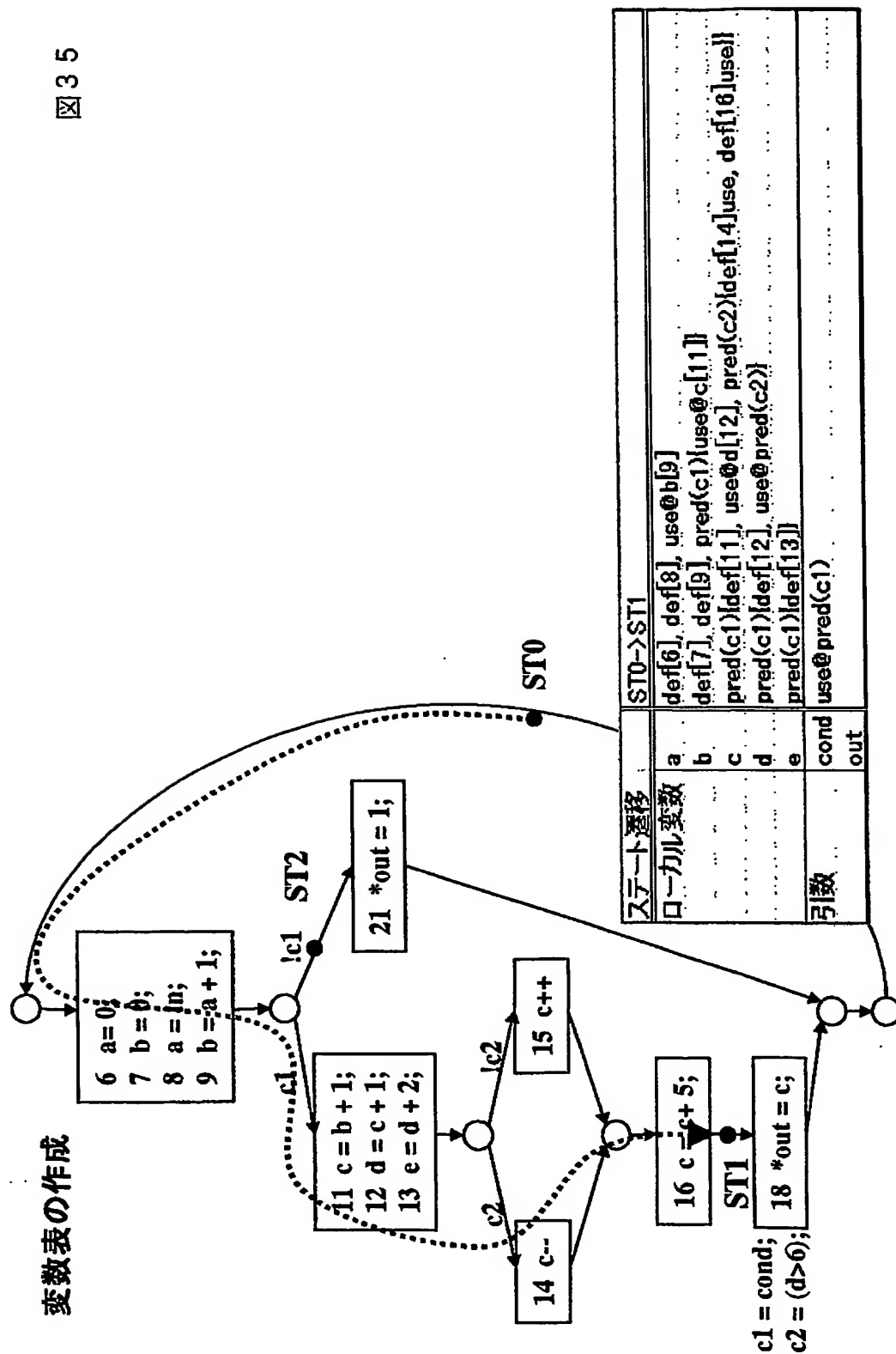
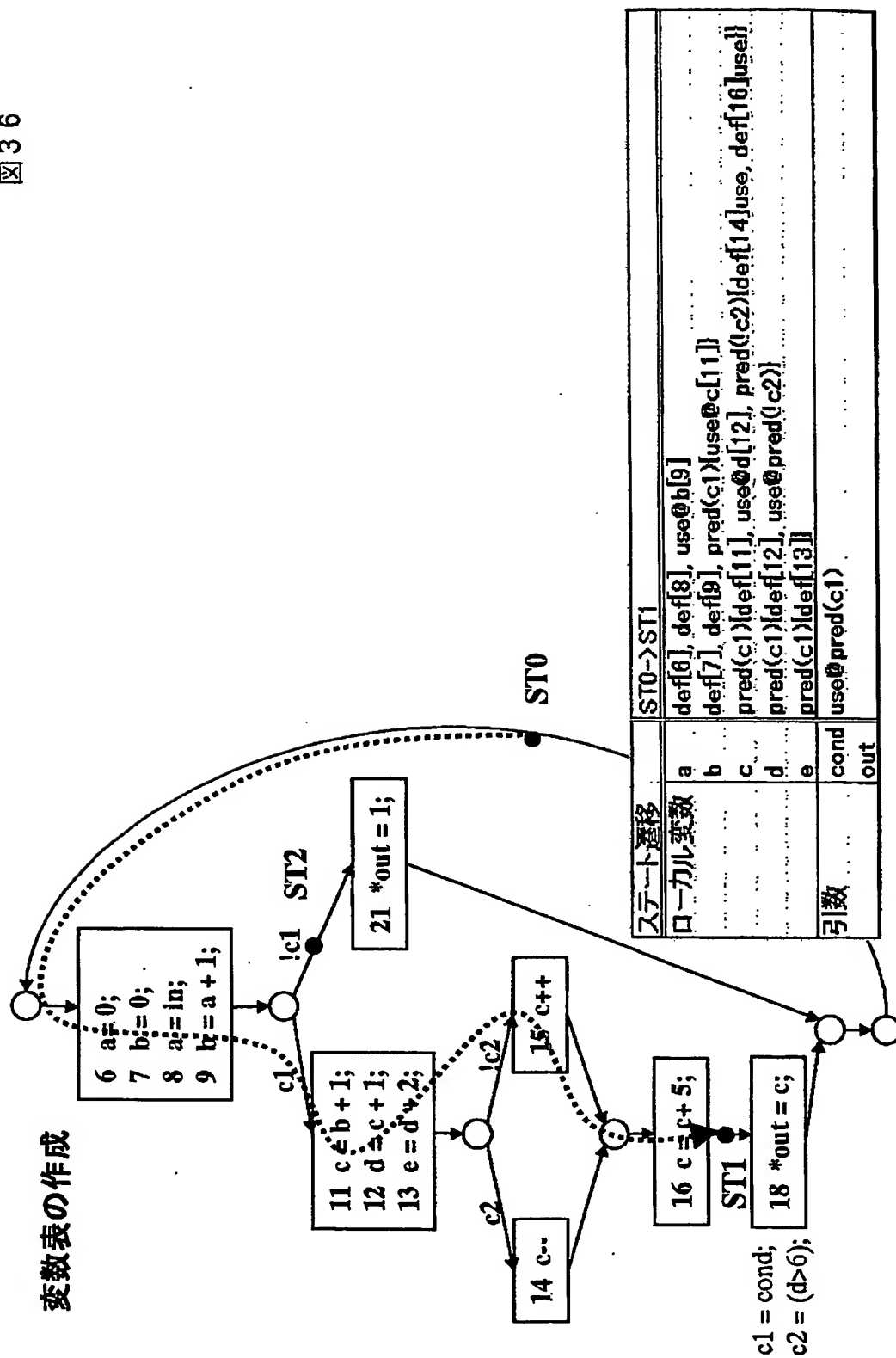


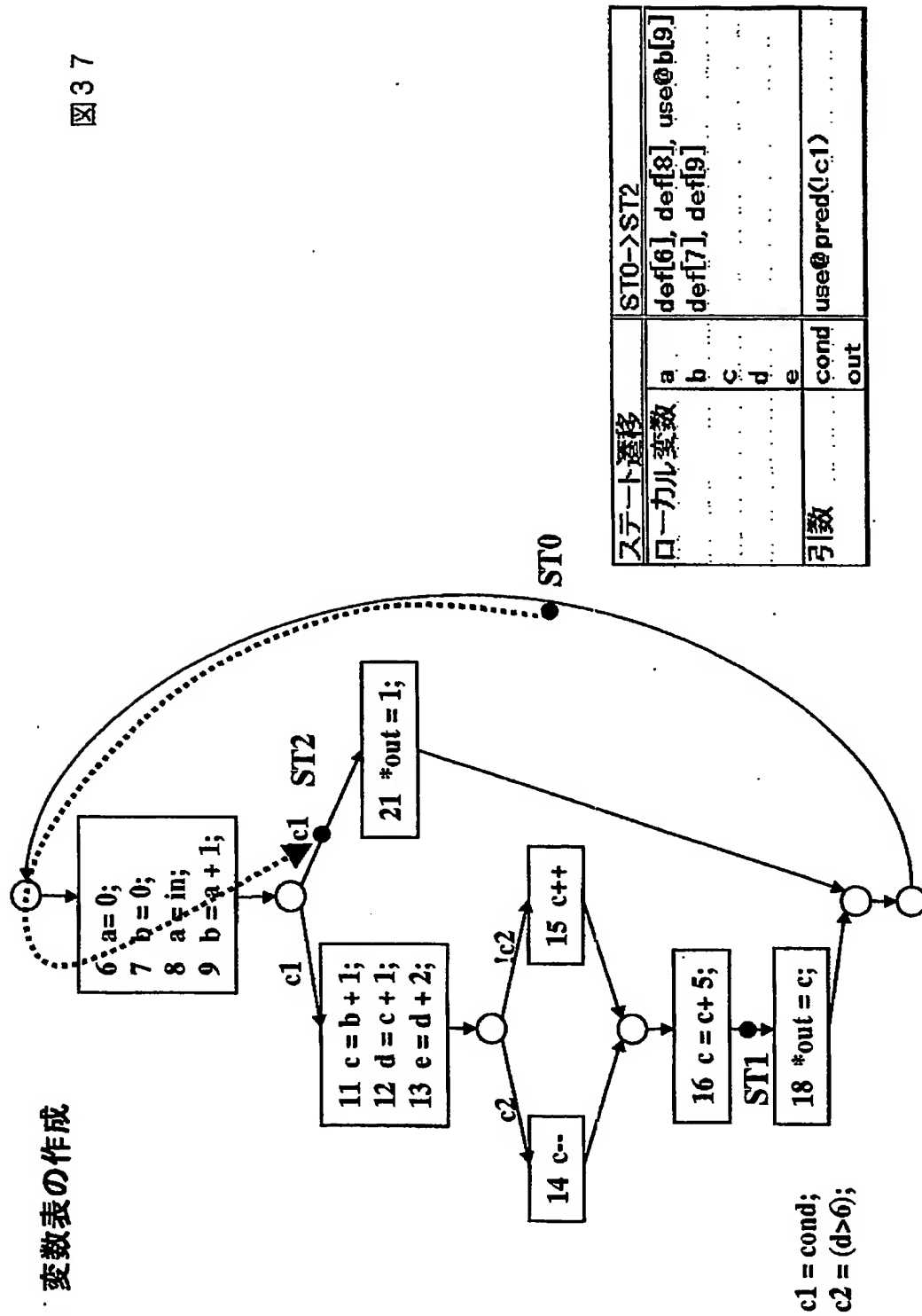
図 36

【図 36】



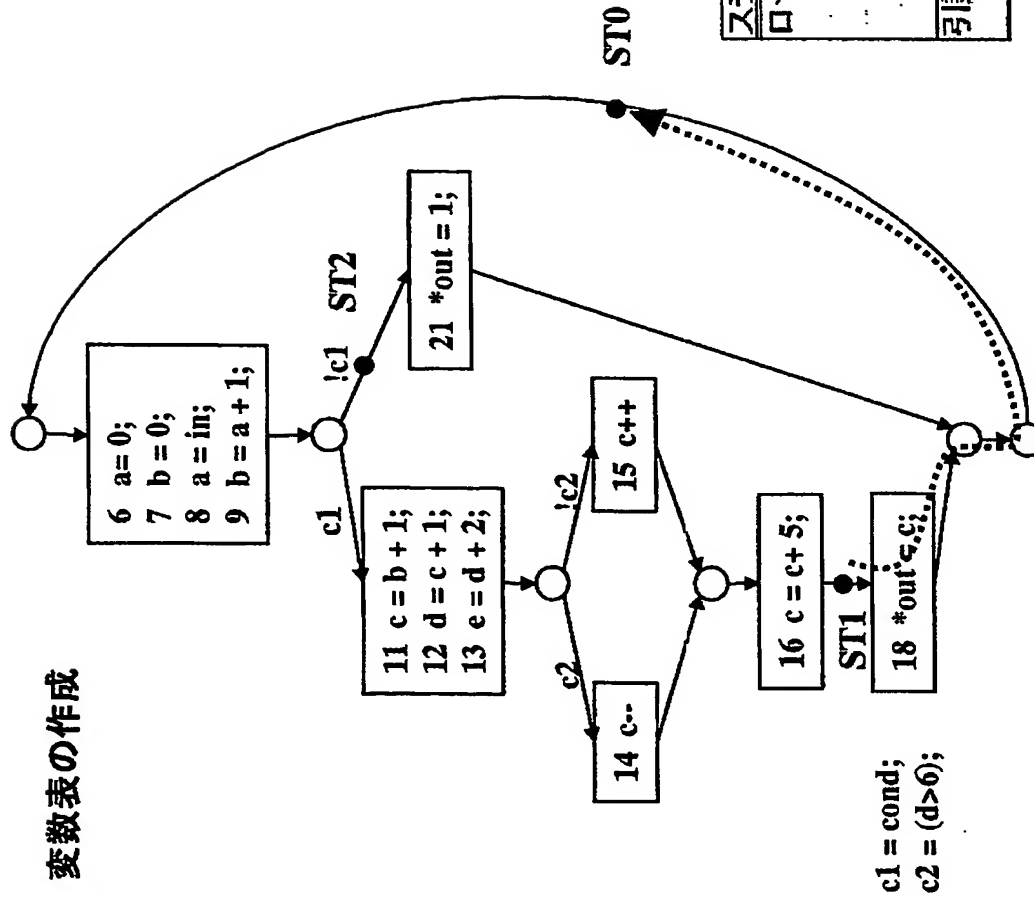
【図 37】

図 37



【図 38】

変数表の作成



ステート遷移		ST1->ST0
ローカル変数	a	
	b	
	c	use@out[17]
	d	
	e	
引数	cond	
	out	def[18]



【図 40】

変数表の作成

図 40

ステート遷移		ST0→ST1	ST0→ST2	ST1→ST0	ST2→ST0
ローカル変数	a	def[6], def[8], use@b[9]	def[6], def[8], use@b[9]		
	b	def[7], def[9], pred(c1)use@c[11]	def[7], def[9]	use@out[17]	
	c	pred(c1)def[11], use@d[12], pred(c2)def[14]use, def[16]use			
	d	pred(c1)def[12], use@pred(c2)			
	e	pred(c1)def[13]			
引数	cond	use@pred(c1)	use@pred(c1)	def[18]	def[21]
	out				
ステート遷移		ST0→ST1			
ローカル変数	a	def[6], def[8], use@b[9]			
	b	def[7], def[9], pred(c1)use@c[11]			
	c	pred(c1)def[11], use@d[12], pred(c2)def[14]use, def[16]use			
	d	pred(c1)def[12], use@pred(c2)			
	e	pred(c1)def[13]			
引数	cond	use@pred(c1)			
	out				

def[n] : n行目で変数定義されている事を表す。
use@var[m] : m行目で変数varへの代入に用いられている事を表す。
pred(cond){...} : 条件condの分岐が成立した場合、{...}が実施される事を表す。
def[]use : l行目で自変数への代入に用いられている事を表す。
use@pred(cond) : 条件condで用いられている事を表す。

【図 4 1】

図 4 1

冗長ステートメント削除

ステート遷移		ST0→ST1	ST0→ST2	ST1→ST0	ST2→ST0
ローカル変数	a	def[6], def[8], use@b[9]	def[6], def[8], use@b[9]		
	b	def[7], def[9], pred(c1)use@c[11]	def[7], def[9]	use@out[17]	
	c	pred(c1)def[11], use@d[12], pred(c2)def[14]use, def[16]use			
	d	pred(c1)def[12], use@pred(c2)			
	e	pred(c1)def[13]			
引数	cond	use@pred(c1)	use@pred(c1)	def[18]	def[21]
	out				
ステート遷移		ST0→ST1			
ローカル変数	a	def[6], def[8], use@b[9]			
	b	def[7], def[9], pred(c1)use@c[11]			
	c	pred(c1)def[11], use@d[12], pred(c2)def[14]use, def[16]use			
	d	pred(c1)def[12], use@pred(c2)			
	e	pred(c1)def[13]			
引数	cond	use@pred(c1)			
	out				

【図 42】

図 42

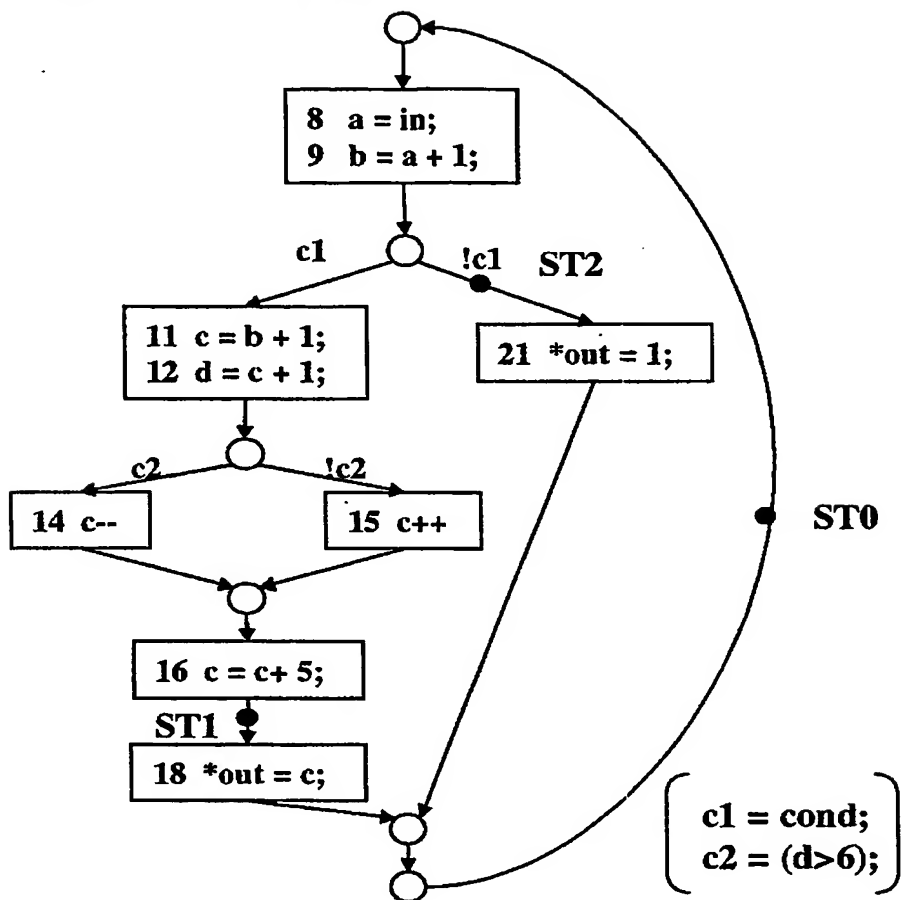
冗長ステートメント削除

ステート遷移		ST0→ST1	ST0→ST2	ST1→ST0	ST2→ST0
ローカル変数	a	def[8], use@b[9]	def[8], use@b[9]		
	b	def[9], pred(c1)use@c[11]			
	c	pred(c1)def[11], use@d[12], pred(c2)def[14]use, def[16]use		use@out[17]	
	d	pred(c1)def[12], use@pred(c2)			
引数	cond	use@pred(c1)	use@pred(c1)	def[18]	def[21]
	out				
ステート遷移		ST0→ST1			
ローカル変数	a	def[8], use@b[9]			
	b	def[9], pred(c1)use@c[11]			
	c	pred(c1)def[11], use@d[12], pred(c2)def[14]use, def[16]use			
	d	pred(c1)def[12], use@pred(c2)			
引数	cond	use@pred(c1)			
	out				

【図 4 3】

図 4 3

冗長ステートメント削除



【図 44】

図 44

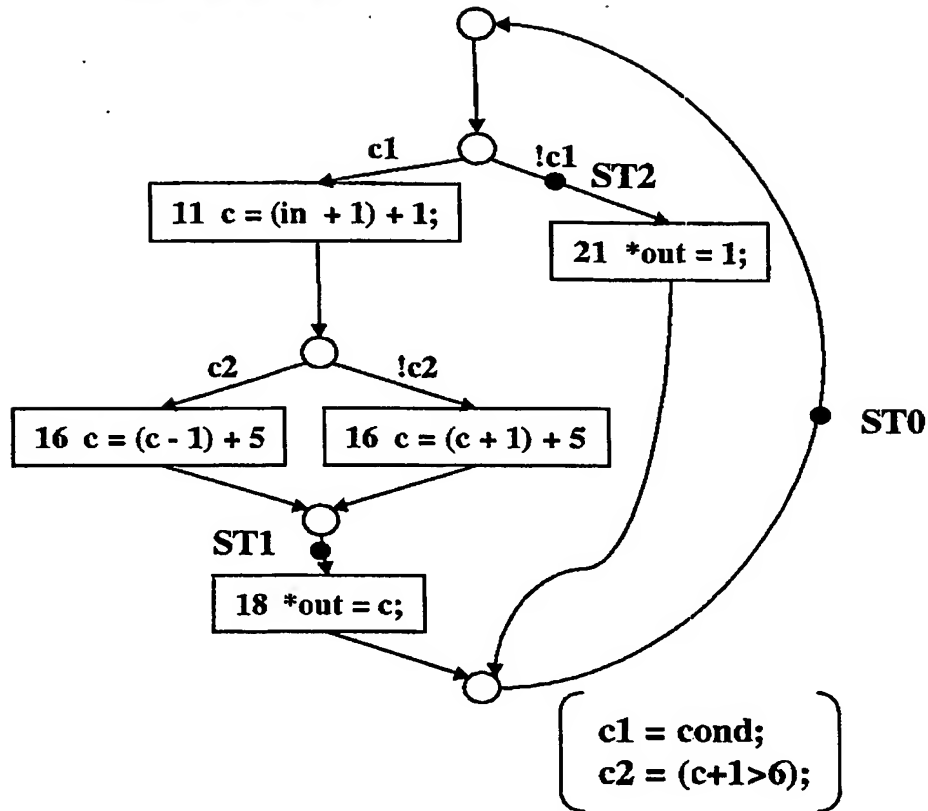
ローカル変数削除

スタート遷移		ST0→ST1	ST0→ST2	ST1→ST0	ST2→ST0
ローカル変数	a	def[8], use@b[9]	def[8], use@b[9]		
	b	def[9], pred(c1)use@c[11]	def[9]		
	c	pred(c1)def[11], use@d[12], pred(c2)def[14]use, def[16]use		use@out[17]	
	d	pred(c1)def[12], use@pred(c2)			
引数	cond	use@pred(c1)	use@pred(lc1)	def[18]	def[21]
	out				
スタート遷移		ST0→ST1			
ローカル変数	a	def[8], use@b[9]			
	b	def[9], pred(c1)use@c[11]			
	c	pred(c1)def[11], use@d[12], pred(lc2)def[14]use, def[16]use			
	d	pred(c1)def[12], use@pred(lc2)			
引数	cond	use@pred(c1)			
	out				

【図 45】

図 45

ローカル変数削除



【図 46】

図 46

更新後の変数表

スタート遷移		ST0→ST1	ST0→ST2	ST1→ST0	ST2→ST0
ローカル変数 引数	c	pred(c1){def[11], pred(c2){def[16]use}}		use@out[17]	
	cond	use@pred(c1)	use@pred(c1)		
	out			def[18]	def[21]
スタート遷移		ST0→ST1			
ローカル変数 引数	c	pred(c1){def[11], pred(c2){def[16]use}}			
	cond	use@pred(c1)			
	out				

【図 47】

図 47

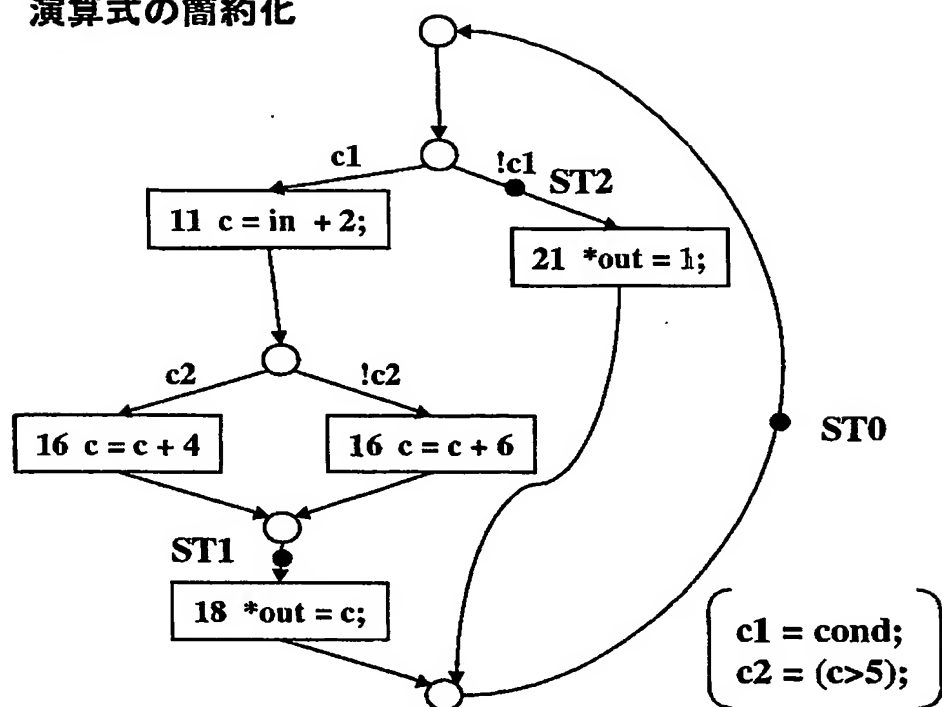
ステート遷移		ST0→ST1	ST0→ST2	ST1→ST0	ST2→ST0
ローカル変数	c	pred(c1)def[11], pred(c2)def[16]use[]	retain	retain	retain
引数	cond	use@pred(c1)	use@pred(lc1)		
	out	retain	retain	def[18]	def[21]
ステート遷移		ST0→ST1			
ローカル変数	c	pred(c1)def[11], pred(lc2)def[16]use[]			
引数	cond	use@pred(c1)			
	out	retain			

[retain : 前保持]

【図 48】

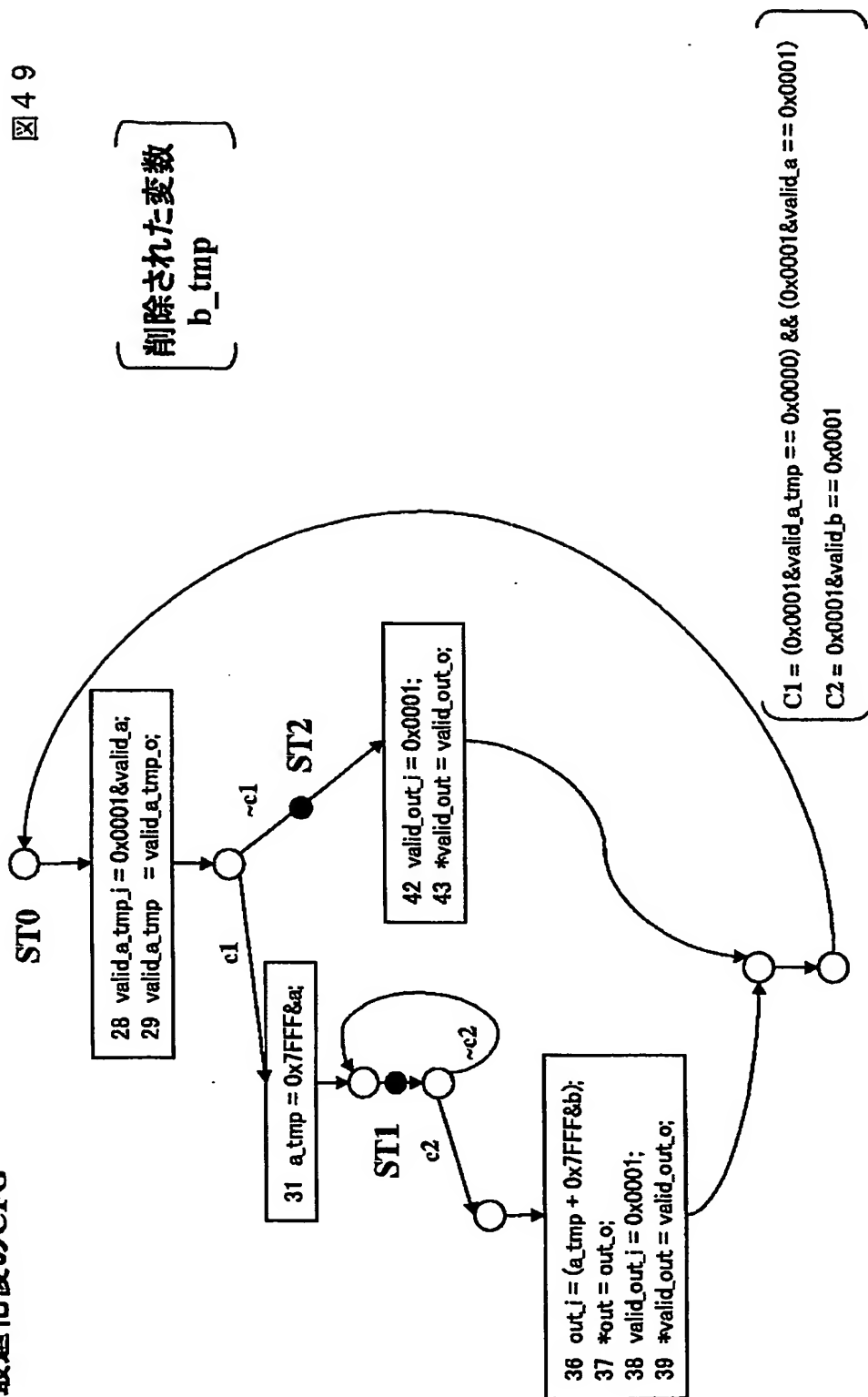
図 48

演算式の簡約化



【図49】

最適化後のCFG



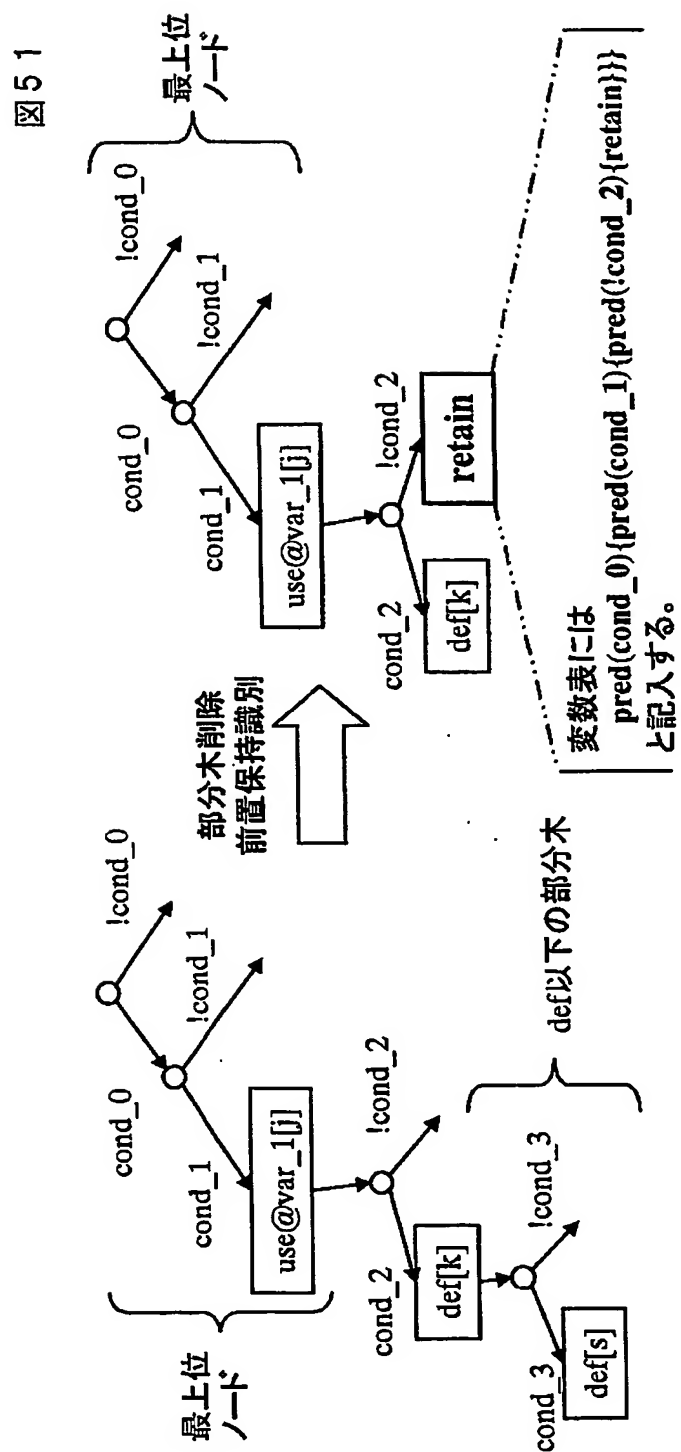
【図 50】

図 50

最適化後の変数表

ステート遷移		ST0→ST1	ST0→ST2	ST1→ST1	ST1→ST2	ST1→ST1
ローカル変数	valid_a tmp	def[29], use@pred(c1)	def[29], use@pred(c1)	pred(c2)def[29], use@pred(c1)	pred(c2)def[29], use@pred(c1)	
	valid_a tmp f	def[28]	def[28]	pred(c2)def[28]	pred(c2)def[28]	
	a tmp	pred(c1)def[31]		pred(c2)def[38]	pred(c2)def[38]	
	out f			pred(c2)def[38]	pred(c2)def[38]	
	valid_out f			pred(c2)def[38]	pred(c2)def[38]	
引数	a	pred(c1)use@a tmp[31]		pred(c2)pred(c1)use@a tmp[31]	pred(c2)pred(c1)use@a tmp[31]	
	b			pred(c2)use@out [38]	pred(c2)use@out [38]	
	valid_a	use@valid_a tmp [28], use@pred(c1)	use@valid_a tmp [28], use@pred(c1)	pred(c2)use@valid_a tmp [28], use@pred(c1)	pred(c2)use@valid_a tmp [28], use@pred(c1)	use@pred(c2)
	valid_b			use@pred(c2)	use@pred(c2)	
	valid_out			pred(c2)def[39]	pred(c2)def[39]	
ステート遷移	out			pred(c2)def[37]	pred(c2)def[37]	
	valid_a tmp_o	use@valid_a tmp[29]	use@valid_a tmp[29]	pred(c2)use@valid_a tmp[29]	pred(c2)use@valid_a tmp[29]	
	valid_out_o			pred(c2)use@valid_out_o[39]	pred(c2)use@valid_out_o[39]	
	out o			pred(c2)use@out[37]	pred(c2)use@out[37]	
ステート遷移		ST1→ST2	ST2→ST1	ST2→ST2	ST2→ST1	ST2→ST2
ローカル変数	valid_a tmp	pred(c2)def[29], use@pred(c1)	def[29], use@pred(c1)	def[29], use@pred(c1)	def[29], use@pred(c1)	
	valid_a tmp f	pred(c2)def[28]	def[28]	pred(c1)def[31]	def[28]	
	a tmp	pred(c2)def[36]		def[42]	def[42]	
	out f	pred(c2)def[38]	def[42]	pred(c1)use@a tmp[31]	pred(c1)use@a tmp[31]	
	valid_out f			use@valid_a tmp [28], use@pred(c1)	use@valid_a tmp [28], use@pred(c1)	
引数	a	pred(c2)use@out [36]		def[43]	def[43]	
	b	pred(c2)use@valid_a tmp [28], use@pred(c1)	use@valid_a tmp [28], use@pred(c1)	use@valid_a tmp [28], use@pred(c1)	use@valid_a tmp [28], use@pred(c1)	
	valid_a	use@pred(c2)		def[43]	def[43]	
	valid_b	pred(c2)def[39]	def[43]	use@valid_out[43]	use@valid_out[43]	
	valid_out	pred(c2)def[37]				
ステート遷移	out	pred(c2)use@valid_a tmp[29]				
	valid_a tmp_o	pred(c2)use@valid_out_o[39]				
	valid_out_o	pred(c2)use@out[37]				
	out o					

【図 51】



【図 52】

図 52

前置保持解析実行後の変数表

ステート遷移	ST0→ST1	ST0→ST2	ST1→ST1	ST1→ST1
ローカル変数				
valid_a, tmp	def[29], use@pred(c1)	def[29], use@pred(c1)	pred(c2)def[29], use@pred(c1)	pred(c2)def[29], use@pred(c1)
valid_a, tmp	def[28]	def[28]	pred(c2)def[28]	pred(c2)def[28]
a, tmp	pred(c1)def[31]	pred(c1)def[31]	pred(c2)pred(c1)def[31]	pred(c2)pred(c1)def[31]
out	retain	retain	pred(c2)def[36]	pred(c2)def[36]
valid_out	retain	retain	pred(c2)def[38]	pred(c2)def[38]
引数				
a	pred(c1)use@a, tmp[31]		pred(c2)pred(c1)use@a, tmp[31]	
b			pred(c2)use@out[38]	
valid_a	use@valid_a, tmp[28], use@pred(c1)	use@valid_a, tmp[28], use@pred(c1)	pred(c2)use@valid_a, tmp[28], use@pred(c1)	use@pred(c2)
valid_b			use@pred(c2)	pred(c2)def[39]
valid_out	retain	retain	pred(c2)def[37]	pred(c2)def[37]
out	retain	retain	pred(c2)use@valid_a, tmp[29]	pred(c2)def[37]
valid_a, tmp_o	use@valid_a, tmp[29]	use@valid_a, tmp[29]	pred(c2)use@valid_out_o[39]	pred(c2)def[37]
valid_out_o			pred(c2)use@out[37]	
out_o				
ステート遷移	ST1→ST2	ST2→ST1	ST2→ST2	ST2→ST2
ローカル変数				
valid_a, tmp	pred(c2)def[29], use@pred(c1)	def[29], use@pred(c1)	def[29], use@pred(c1)	
valid_a, tmp	pred(c2)def[28]	def[28]	def[28]	
a, tmp	pred(c1)def[31]	pred(c1)def[31]	pred(c1)def[31]	
out	retain	retain	retain	
valid_out	pred(c2)def[38]	def[42]	def[42]	
引数				
a	pred(c2)use@out[38]	pred(c1)use@a, tmp[31]		
b	pred(c2)use@valid_a, tmp[28], use@pred(c1)	use@valid_a, tmp[28], use@pred(c1)	use@valid_a, tmp[28], use@pred(c1)	
valid_a	use@pred(c2)			
valid_b	pred(c2)def[39]	def[43]	def[43]	
valid_out	pred(c2)def[37]	retain	retain	
out	pred(c2)use@valid_a, tmp[29]	use@valid_out[43]	use@valid_out[43]	
valid_a, tmp_o	pred(c2)use@valid_out_o[39]			
valid_out_o				
out_o	pred(c2)use@out[37]			

【図 53】

図 53

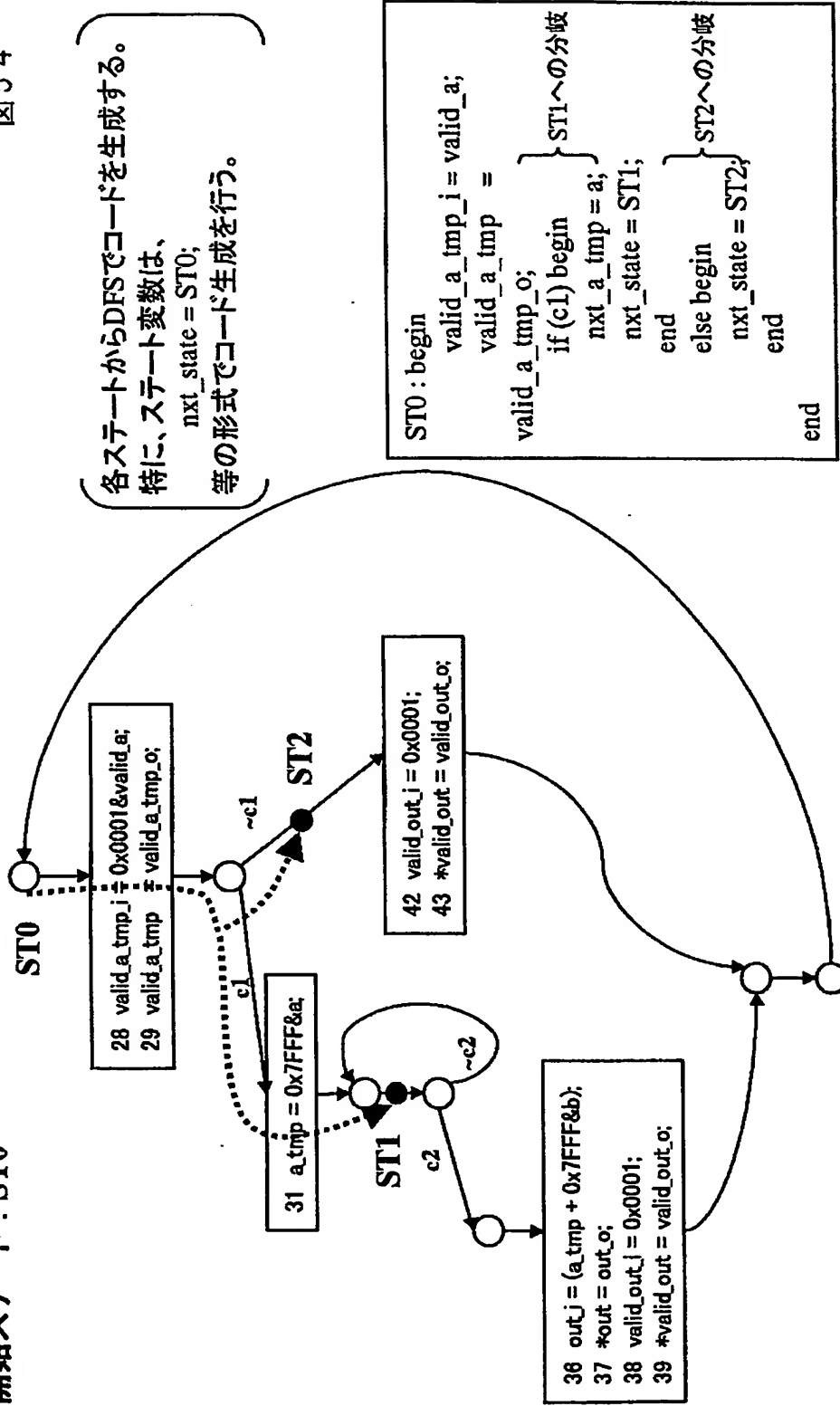
前置保持解析結果の変数表からの情報取得

変数	ST1-ST1	ST2-ST2	ST1-ST1	ST1-ST1
ローカル変数				
valid a tmp	def[29], use@pred(c1)	def[29], use@pred(c1)	pred(c2)def[29], use@pred(c1)	pred(c2)valid a tmp = valid a tmp o;
valid a tmp i	def[28]	def[28]	pred(c2)def[28]	pred(c2)valid a tmp i = valid a tmp o;
a tmp	pred(c1)def[31]	pred(c1)next a tmp = tmp	pred(c2)pred(c1)def[31]	pred(c2)next a tmp = a tmp
out i	out i = out o;	out i = out o;	pred(c2)def[33]	pred(c2)out i = out o;
valid out i	valid out i = valid out o;	valid out i = valid out o;	pred(c2)def[38]	pred(c2)valid out i = valid out o;
引数				
a	pred(c1)use@a tmp[31]	pred(c1)use@a tmp[31]	pred(c2)pred(c1)use@a tmp[31]	
b	use@valid a tmp[28], use@pred(c1)	use@valid a tmp[28], use@pred(c1)	pred(c2)use@out[36]	
valid a	valid out = valid out o;	valid out = valid out o;	pred(c2)def[39]	use@pred(c2)
valid out	out = out o;	out = out o;	pred(c2)def[37]	pred(c2)valid out = valid out o;
valid a tmp o	use@valid a tmp[29]	use@valid a tmp[29]	pred(c2)use@valid a tmp[29]	pred(c2)out = out o;
valid out o			pred(c2)use@out[37]	
out o				
グローバル変数				
valid a tmp	ST1-ST2	ST2-ST1	ST2-ST2	
valid a tmp i	pred(c2)def[29], use@pred(c1)	def[29], use@pred(c1)	def[29], use@pred(c1)	
a tmp	pred(c2)def[28]	def[28]	def[28]	
out i	pred(c1)next a tmp = a tmp	pred(c1)def[31]	pred(c1)next a tmp = a tmp	
valid out i	pred(c2)def[36]	out i = out o;	out i = out o;	
	pred(c2)def[38]	def[42]	def[42]	
引数				
a	pred(c2)use@out[36]	pred(c1)use@a tmp[31]		
b	pred(c2)use@valid a tmp[28], use@pred(c1)	use@valid a tmp[28], use@pred(c1)	use@valid a tmp[28], use@pred(c1)	
valid a	use@pred(c2)			
valid out	pred(c2)def[39]	def[43]	def[43]	
out	pred(c2)def[37]	out = out o;	out = out o;	
valid a tmp o	pred(c2)use@valid a tmp[29]			
valid out o	pred(c2)use@valid out o[38]	use@valid out[43]	use@valid out[43]	
out o	pred(c2)use@out[37]			

【図 54】

図 54

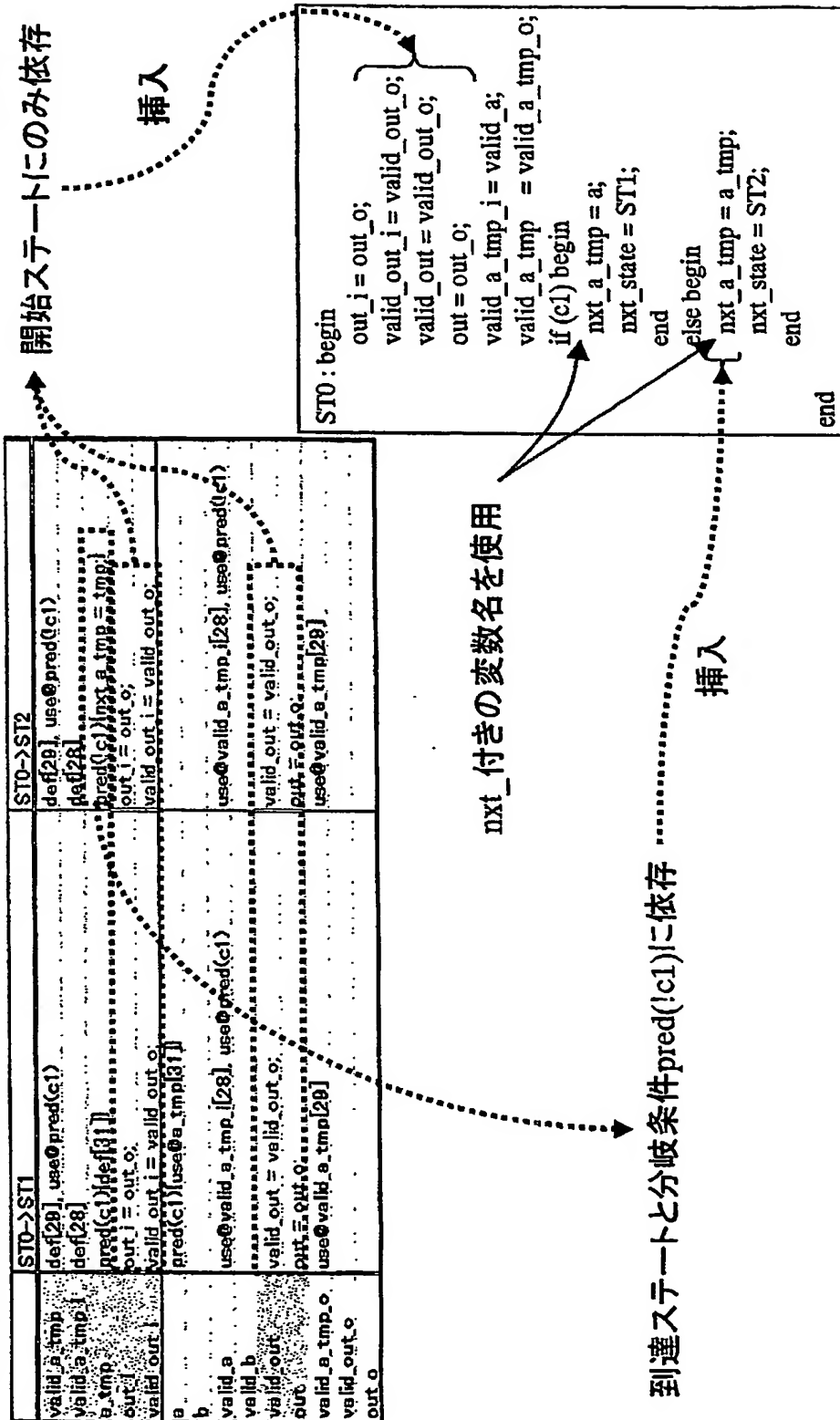
開始ステート: ST0



【図 55】

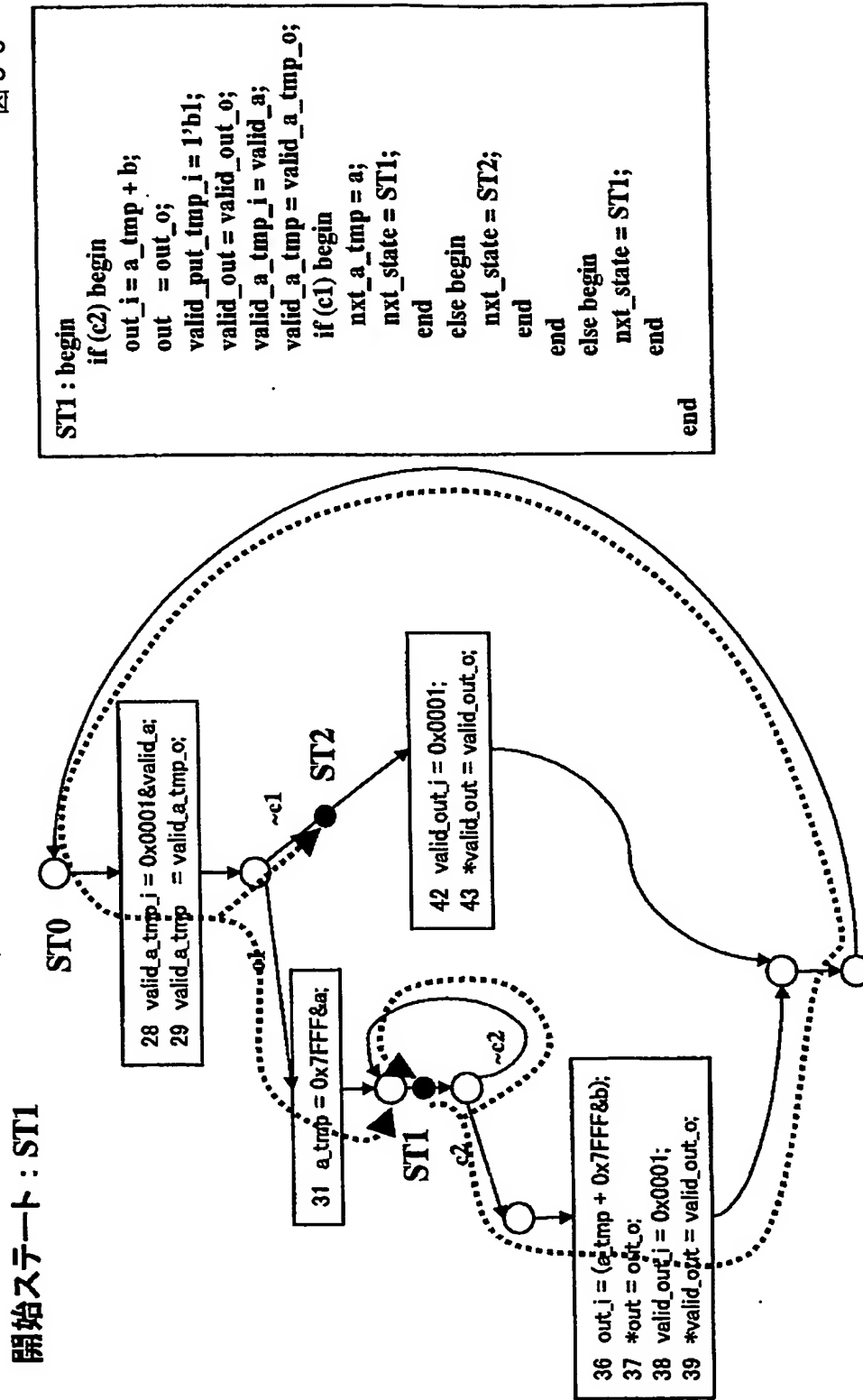
図 55

開始ステート: ST0



【図 56】

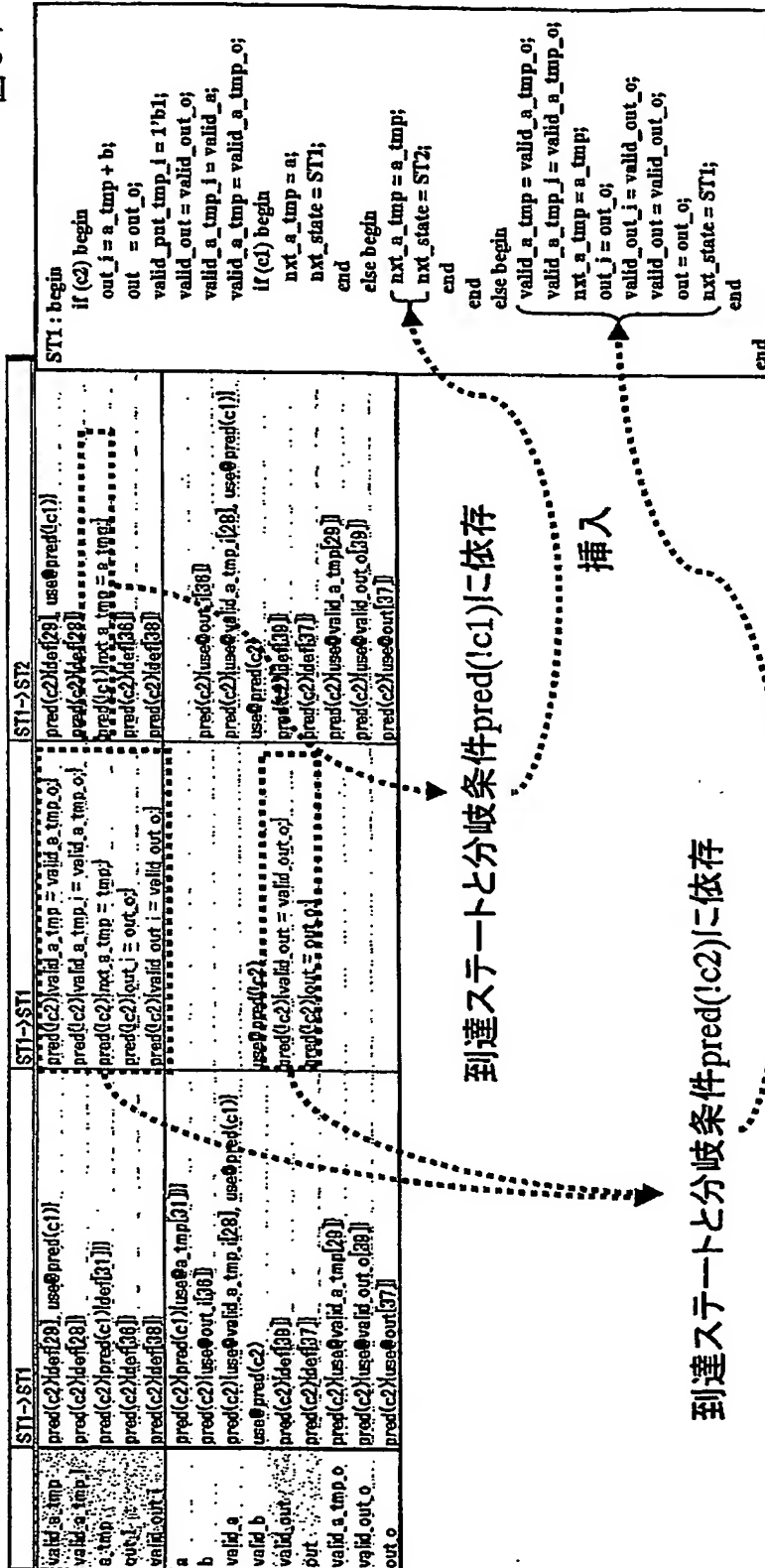
図 56



【図 57】

図 57

開始ステート: ST1



到達ステートと分岐条件pred(c1)に依存

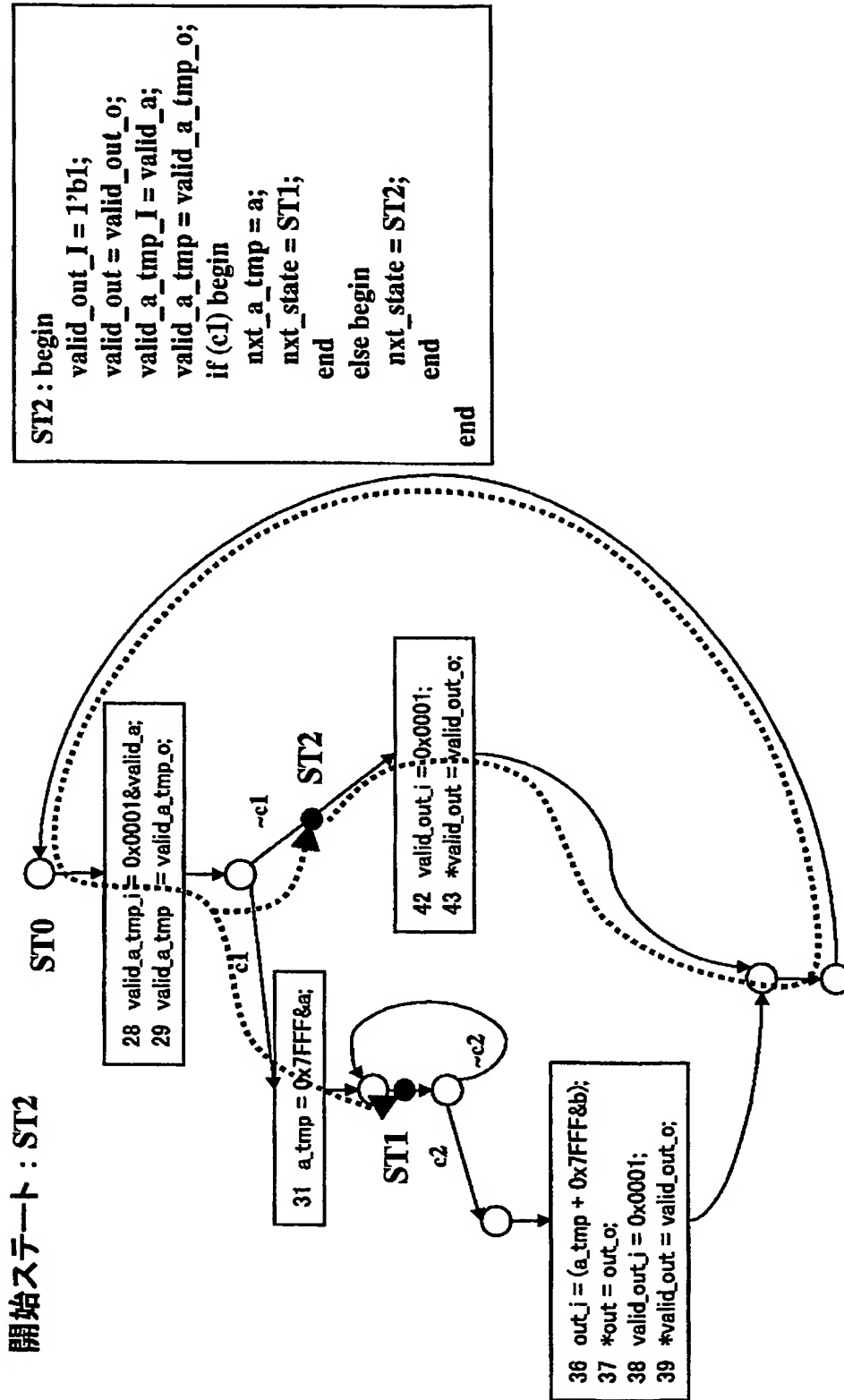
挿入

到達ステートと分岐条件pred(c2)に依存

挿入

【図 58】

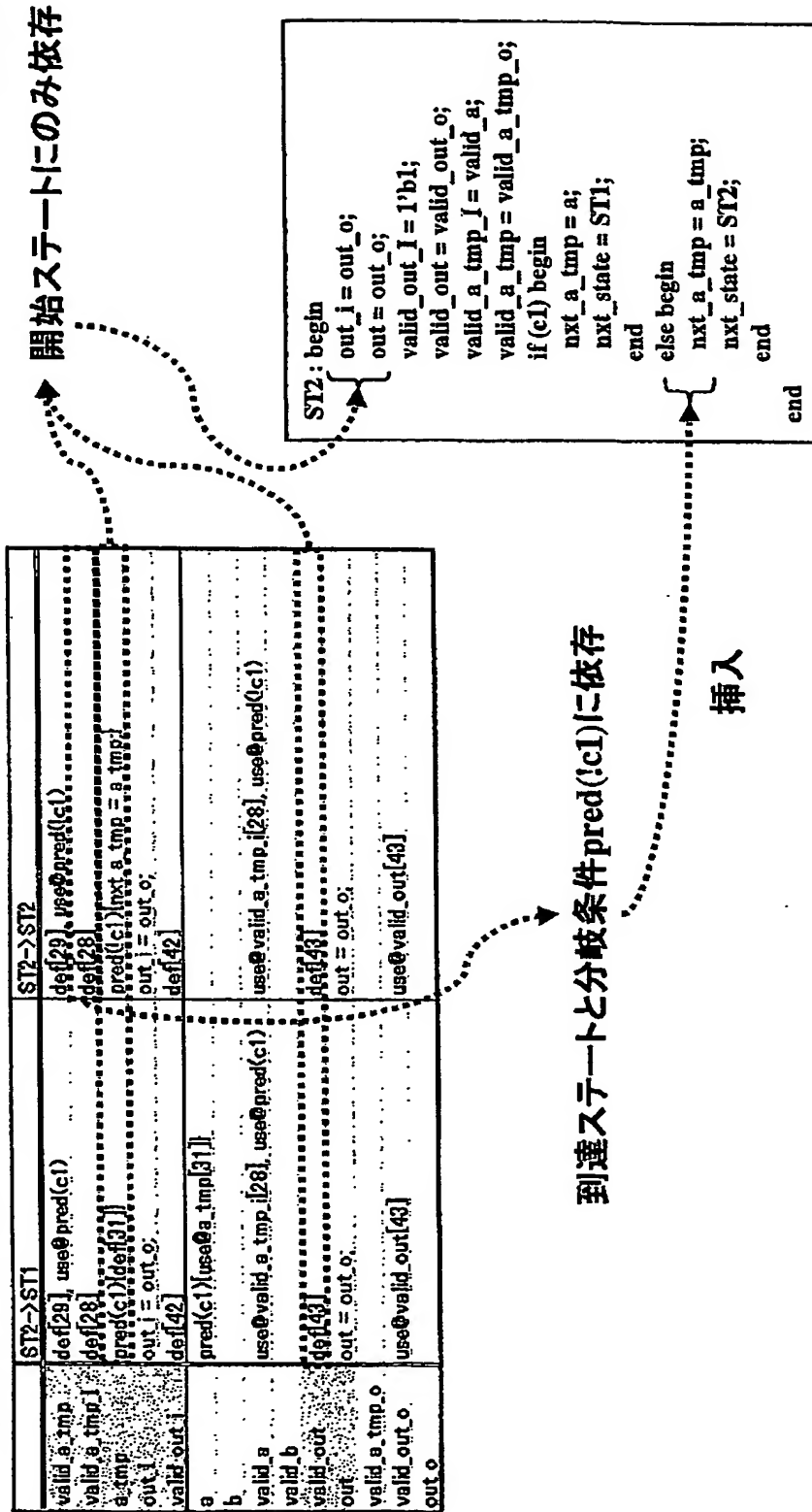
図 58



【図 59】

図 5-9

開始状態: ST2



【図 60】

図 60

```

1  module PipeLine(clk, reset_n,
2      valid_a, valid_b, a, b,
3      out, valid_out);
4      // System clock and reset
5      input clk;
6      input reset_n;
7      // PipeLine input signals
8      input valid_a;
9      input valid_b;
10     input [14:0] a;
11     input [14:0] b;
12     // PipeLine output signals
13     output valid_out;
14     reg valid_out;
15     reg valid_a_tmp_i;
16     reg valid_a_tmp_o;
17     reg [14:0] a_tmp;
18     reg [14:0] nxt_a_tmp;
19     reg valid_out_i;
20     reg valid_out_o;
21     reg [15:0] out_i;
22     reg [15:0] out_o;
23     // State registers
24     reg [1:0] state, nxt_state;
25     parameter ST0=2'b00,
26               ST1=2'b01,
27               ST2=2'b10;
28     // Blanch conditions
29     wire c1;
30     wire c2;
31     assign c1 = !valid_a_tmp&&valid_a;
32     assign c2 = valid_b;

```

【図 61】

図 61

```

// Mealy finite state machine
always @ (state or c1 or c2 or
    valid_a_tmp_i or valid_a_tmp_o or
    valid_a_tmp or a_tmp or
    valid_out_i or valid_out_o or
    out_i or out_o) begin
    case(state[1:0])
    ST0 : begin
        valid_a_tmp_j = valid_a;
        valid_a_tmp = valid_a_tmp_o;
        valid_out_j = valid_out_o;
        valid_out = valid_out_o;
        out_j = out_o;
        out = out_o;
        if (c1) begin
            nxt_a_tmp = a;
            nxt_state = ST1;
        end
    else begin
        nxt_a_tmp = a_tmp;
        nxt_state = ST2;
    end
    end
end

51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72

// Register assignment statement
always @ (posedge clk or negedge reset_n) begin
    if (!reset_n) begin
        valid_a_tmp_o <= 1'b0;
        out_o <= 17'b00000000000000000000;
    end
    else begin
        valid_a_tmp_o <= valid_a_tmp_i;
        out_o <= out_i;
    end
end

// State registers and temporal registers
always @ (posedge clk or negedge reset_n) begin
    if (!reset_n) begin
        state <= ST0;
        a_tmp <= 16'b0;
    end
    else begin
        state <= nxt_state;
        a_tmp <= nxt_a_tmp;
    end
end

```

【図 6 2】

図 6 2

```

72      ST1 : begin
73      if (c2) begin
74          out_j      = a_tmp + b;
75          out         = out_o;
76          valid_out_j = 1'b1;
77          valid_out   = valid_out_o;
78          valid_a_tmp_j = valid_a;
79          valid_a_tmp = valid_a_tmp_o;
80          if (c1) begin
81              nxt_a_tmp = a;
82              nxt_state = ST1;
83          end
84          else begin
85              nxt_a_tmp = a_tmp;
86              nxt_state = ST2;
87          end
88          end
89          else begin
90              nxt_state = ST1;
91              valid_a_tmp_j = valid_a_tmp_o;
92              valid_a_tmp = valid_a_tmp_o;
93              nxt_a_tmp   = a_tmp;
94              valid_out_j = valid_out_o;
95              valid_out   = valid_out_o;
96              out_j       = out_o;
97              out         = out_o;
98          end
99      end
100  end

101  ST2 : begin
102      valid_a_tmp_j = valid_a;
103      valid_a_tmp = valid_a_tmp_o;
104      valid_out_j = 1'b0;
105      valid_out   = valid_out_o;
106      out_j       = out_o;
107      out         = out_o;
108      if (c1) begin
109          nxt_a_tmp = a;
110          nxt_state = ST1;
111      end
112      else begin
113          nxt_a_tmp = a_tmp;
114          nxt_state = ST2;
115      end
116  end
117  default : begin
118      nxt_state = ST0;
119      valid_a_tmp_j = valid_a_tmp_o;
120      valid_a_tmp = 1'b0;
121      nxt_a_tmp   = 15'b0;
122      valid_out_j = valid_out_o;
123      valid_out   = 1'b0;
124      out_j       = out_o;
125      out         = out_o;
126  end
127  endcase
128  end
129 endmodule

```

【書類名】 要約書

【要約】

【課題】 ストール動作を伴うパイプライン動作が可能な回路のプログラム記述又は回路記述を容易に得る事ができるコンパイラを提供する。

【解決手段】 クロック境界及びレジスタ代入文によりステートメントレベルでの並列動作の記述をサイクル精度で記述可能な擬似C記述(1)を入力とし、レジスタ代入文の識別を行い(S2)、実行可能なC記述(3)を生成すると共に、状態数削減を行ったステートマシンを抽出し、0サイクルで実行されるループが存在するか否かを判定し(S5)、もしなければ、論理合成可能な回路記述(4)を生成する。クロック境界を明示的にC記述内に挿入した擬似C記述を入力し、レジスタ代入文によるステートメントレベルでの並列記述を可能にした擬似C記述を入力するから、ストール動作を伴うパイプライン動作が表現可能である。

【選択図】 図1

【書類名】 出願人名義変更届（一般承継）

【あて先】 特許庁長官 殿

【事件の表示】

【出願番号】 特願2002-300073

【承継人】

【識別番号】 503121103

【氏名又は名称】 株式会社ルネサステクノロジ

【承継人代理人】

【識別番号】 100089071

【弁理士】

【氏名又は名称】 玉村 静世

【提出物件の目録】

【包括委任状番号】 0308734

【物件名】 承継人であることを証明する登記簿謄本 1

【援用の表示】 特許第 3 1 5 4 5 4 2 号 平成 1 5 年 4 月 1 1 日付け
提出の会社分割による特許権移転登録申請書 を援用
する

【物件名】 権利の承継を証明する承継証明書 1

【援用の表示】 特願平 2 - 3 2 1 6 4 9 号 同日提出の出願人
名義変更届（一般承継）を援用する

【プルーフの要否】 要

認定・付加情報

特許出願の番号	特願 2002-300073
受付番号	50301210809
書類名	出願人名義変更届 (一般承継)
担当官	末武 実 1912
作成日	平成15年10月 7日

<認定情報・付加情報>

【提出日】	平成15年 7月23日
-------	-------------

特願 2002-300073

出願人履歴情報

識別番号

[000005108]

1. 変更年月日

1990年 8月31日

[変更理由]

新規登録

住 所

東京都千代田区神田駿河台4丁目6番地

氏 名

株式会社日立製作所

特願 2 0 0 2 - 3 0 0 0 7 3

出 願 人 履 歴 情 報

識別番号

[5 0 3 1 2 1 1 0 3]

1. 変 更 年 月 日

2 0 0 3 年 4 月 1 日

[変 更 理 由]

新 規 登 録

住 所

東 京 都 千 代 田 区 丸 の 内 二 丁 目 4 番 1 号

氏 名

株 式 会 社 ル ネ サ ス テ ク ノ ロ ジ

Translation

PATENT COOPERATION TREATY

PCT

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

(PCT Article 36 and Rule 70)

PCT/JP2003/012839



Applicant's or agent's file reference 310201409WO1	FOR FURTHER ACTION See Notification of Transmittal of International Preliminary Examination Report (Form PCT/IPEA/416)	
International application No. PCT/JP2003/012839	International filing date (day/month/year) 07 October 2003 (07.10.2003)	Priority date (day/month/year) 15 October 2002 (15.10.2002)
International Patent Classification (IPC) or national classification and IPC G06F 17/50		
Applicant RENESAS TECHNOLOGY CORP.		

1. This international preliminary examination report has been prepared by this International Preliminary Examining Authority and is transmitted to the applicant according to Article 36.

2. This REPORT consists of a total of 4 sheets, including this cover sheet.

☒ This report is also accompanied by ANNEXES, i.e., sheets of the description, claims and/or drawings which have been amended and are the basis for this report and/or sheets containing rectifications made before this Authority (see Rule 70.16 and Section 607 of the Administrative Instructions under the PCT).

These annexes consist of a total of 3 sheets.

3. This report contains indications relating to the following items:

- I ☒ Basis of the report
- II ☐ Priority
- III ☐ Non-establishment of opinion with regard to novelty, inventive step and industrial applicability
- IV ☐ Lack of unity of invention
- V ☒ Reasoned statement under Article 35(2) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement
- VI ☐ Certain documents cited
- VII ☐ Certain defects in the international application
- VIII ☒ Certain observations on the international application

Date of submission of the demand 07 October 2003 (07.10.2003)	Date of completion of this report 27 January 2004 (27.01.2004)
Name and mailing address of the IPEA/JP	Authorized officer
Facsimile No.	Telephone No.

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

International application No.

PCT/JP2003/012839

I. Basis of the report

1. With regard to the elements of the international application:*

- ☐ the international application as originally filed
- ☒ the description:
 pages 1-35, as originally filed
 pages _____, filed with the demand
 pages _____, filed with the letter of _____
- ☒ the claims:
 pages 4,5, 7-13, 15-20, as originally filed
 pages _____, as amended (together with any statement under Article 19
 pages _____, filed with the demand
 pages 1-3, 6, 14, filed with the letter of 19 January 2004 (19.01.2004)
- ☒ the drawings:
 pages _____, as originally filed
 pages _____, filed with the demand
 pages _____, filed with the letter of _____
- ☐ the sequence listing part of the description:
 pages _____, as originally filed
 pages _____, filed with the demand
 pages _____, filed with the letter of _____

2. With regard to the language, all the elements marked above were available or furnished to this Authority in the language in which the international application was filed, unless otherwise indicated under this item.

These elements were available or furnished to this Authority in the following language _____ which is:

- ☐ the language of a translation furnished for the purposes of international search (under Rule 23.1(b)).
- ☐ the language of publication of the international application (under Rule 48.3(b)).
- ☐ the language of the translation furnished for the purposes of international preliminary examination (under Rule 55.2 and/or 55.3).

3. With regard to any nucleotide and/or amino acid sequence disclosed in the international application, the international preliminary examination was carried out on the basis of the sequence listing:

- ☐ contained in the international application in written form.
- ☐ filed together with the international application in computer readable form.
- ☐ furnished subsequently to this Authority in written form.
- ☐ furnished subsequently to this Authority in computer readable form.
- ☐ The statement that the subsequently furnished written sequence listing does not go beyond the disclosure in the international application as filed has been furnished.
- ☐ The statement that the information recorded in computer readable form is identical to the written sequence listing has been furnished.

4. ☐ The amendments have resulted in the cancellation of:

- ☐ the description, pages _____
- ☐ the claims, Nos. _____
- ☐ the drawings, sheets/fig _____

5. ☐ This report has been established as if (some of) the amendments had not been made, since they have been considered to go beyond the disclosure as filed, as indicated in the Supplemental Box (Rule 70.2(c)).**

* Replacement sheets which have been furnished to the receiving Office in response to an invitation under Article 14 are referred to in this report as "originally filed" and are not annexed to this report since they do not contain amendments (Rule 70.16 and 70.17).

** Any replacement sheet containing such amendments must be referred to under item 1 and annexed to this report.

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

International application No.

PCT/JP03/12839

V. Reasoned statement under Article 35(2) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

1. Statement

Novelty (N)	Claims	2, 3, 7-20	YES
	Claims	1, 4-6	NO
Inventive step (IS)	Claims	15-20	YES
	Claims	1-14	NO
Industrial applicability (IA)	Claims	1-20	YES
	Claims		NO

2. Citations and explanations

Document 1: KAZUTOSHI WAKABAYASHI, ET AL., "Densoyo LSI o Dosa Gosei de Kaihatsu, Kino Sekkei no Kikan ga 1/10 ni Tanshuku," Nikkei Electronics, Nikkei Business Publications, Inc., 12 February 1996 (12.02.96), No. 655, pages 147-169

Document 2: JP, 2002-49652, A (HIROSHI YASUDA), February 15, 2002 (02.15. 02.02), claims 1-5 (Family: none)

The Cyber described in document 1 performs high-level synthesis based on the operation description language BDL (with grammar similar to C language) and outputs RTL description. The aforesaid BDL can describe by changing the clock change point (i.e. the clock boundary) to "\$".

Here, when inserting to a variable between \$ and \$, it is unclear to what register the variable is allocated, but insertion to a register occurs, so it could be called a register insertion statement.

Furthermore, the "=" and "++" in document 1 could be called "specific operators."

Also, document 2 discloses something equivalent to a second program description.

Meanwhile, compiling something described in one language to a description using another language is well-known art.

This being the case, compiling the aforesaid BDL to the description described in document 2 is merely something to be appropriately achieved by a person skilled in the art.

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

International application No.

PCT/JP03/12839

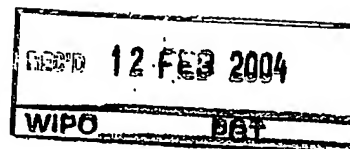
VIII. Certain observations on the international application

The following observations on the clarity of the claims, description, and drawings or on the question whether the claims are fully supported by the description, are made:

The technical scope of "register insertion statement" in claims 1 through 20 is unclear.

PCT

国際予備審査報告

(法第12条、法施行規則第56条)
[PCT36条及びPCT規則70]

出願人又は代理人 の書類記号 310201409WO1	今後の手続きについては、国際予備審査報告の送付通知(様式PCT/ IPEA/416)を参照すること。	
国際出願番号 PCT/JPO3/12839	国際出願日 (日.月.年) 07.10.03	優先日 (日.月.年) 15.10.02
国際特許分類 (IPC) Int. Cl. 7 G06F17/50		
出願人 (氏名又は名称) 株式会社ルネサステクノロジ		

1. 国際予備審査機関が作成したこの国際予備審査報告を法施行規則第57条 (PCT36条) の規定に従い送付する。

2. この国際予備審査報告は、この表紙を含めて全部で 4 ページからなる。

☒ この国際予備審査報告には、附属書類、つまり補正されて、この報告の基礎とされた及び/又はこの国際予備審査機関に対してした訂正を含む明細書、請求の範囲及び/又は図面も添付されている。
(PCT規則70.16及びPCT実施細則第607号参照)
この附属書類は、全部で 3 ページである。

3. この国際予備審査報告は、次の内容を含む。

I ☒ 国際予備審査報告の基礎

II ☐ 優先権

III ☐ 新規性、進歩性又は産業上の利用可能性についての国際予備審査報告の不作成

IV ☐ 発明の単一性の欠如

V ☒ PCT35条(2)に規定する新規性、進歩性又は産業上の利用可能性についての見解、それを裏付けるための文献及び説明

VI ☐ ある種の引用文献

VII ☐ 国際出願の不備

VIII ☒ 国際出願に対する意見

国際予備審査の請求書を受理した日 07.10.03	国際予備審査報告を作成した日 27.01.04	
名称及びあて先 日本国特許庁 (IPEA/JP) 郵便番号100-8915 東京都千代田区霞が関三丁目4番3号	特許庁審査官 (権限のある職員) 早川 学 電話番号 03-3581-1101 内線 3531	5H 9652

様式PCT/IPEA/409 (表紙) (1998年7月)

I. 国際予備審査報告の基礎

1. この国際予備審査報告は下記の出願書類に基づいて作成された。(法第6条(PCT14条)の規定に基づく命令に
応答するために提出された差し替え用紙は、この報告書において「出願時」とし、本報告書には添付しない。
PCT規則70.16, 70.17)

☐ 出願時の国際出願書類

- ☒ 明細書 第 1-35 ページ、 出願時に提出されたもの
明細書 第 _____ ページ、 国際予備審査の請求書と共に提出されたもの
明細書 第 _____ ページ、 _____ 付の書簡と共に提出されたもの

- ☒ 請求の範囲 第 4, 5, 7-13, 15-20 項、 出願時に提出されたもの
請求の範囲 第 _____ 項、 PCT19条の規定に基づき補正されたもの
請求の範囲 第 _____ 項、 国際予備審査の請求書と共に提出されたもの
請求の範囲 第 1-3, 6, 14 項、 19.01.04 付の書簡と共に提出されたもの

- ☒ 図面 第 1-62 ~~ページ~~/図、 出願時に提出されたもの
図面 第 _____ ページ/図、 国際予備審査の請求書と共に提出されたもの
図面 第 _____ ページ/図、 _____ 付の書簡と共に提出されたもの

- ☐ 明細書の配列表の部分 第 _____ ページ、 出願時に提出されたもの
明細書の配列表の部分 第 _____ ページ、 国際予備審査の請求書と共に提出されたもの
明細書の配列表の部分 第 _____ ページ、 _____ 付の書簡と共に提出されたもの

2. 上記の出願書類の言語は、下記に示す場合を除くほか、この国際出願の言語である。

上記の書類は、下記の言語である _____ 語である。

- ☐ 国際調査のために提出されたPCT規則23.1(b)にいう翻訳文の言語
☐ PCT規則48.3(b)にいう国際公開の言語
☐ 国際予備審査のために提出されたPCT規則55.2または55.3にいう翻訳文の言語

3. この国際出願は、ヌクレオチド又はアミノ酸配列を含んでおり、次の配列表に基づき国際予備審査報告を行った。

- ☐ この国際出願に含まれる書面による配列表
☐ この国際出願と共に提出された磁気ディスクによる配列表
☐ 出願後に、この国際予備審査(または調査)機関に提出された書面による配列表
☐ 出願後に、この国際予備審査(または調査)機関に提出された磁気ディスクによる配列表
☐ 出願後に提出した書面による配列表が出願時における国際出願の開示の範囲を超える事項を含まない旨の陳述書の提出があった
☐ 書面による配列表に記載した配列と磁気ディスクによる配列表に記載した配列が同一である旨の陳述書の提出があった。

4. 補正により、下記の書類が削除された。

- ☐ 明細書 第 _____ ページ
☐ 請求の範囲 第 _____ 項
☐ 図面 図面の第 _____ ページ/図

5. ☐ この国際予備審査報告は、補充欄に示したように、補正が出願時における開示の範囲を越えてされたものと認められるので、その補正がされなかったものとして作成した。(PCT規則70.2(c) この補正を含む差し替え用紙は上記1.における判断の際に考慮しなければならない、本報告に添付する。)

V. 新規性、進歩性又は産業上の利用可能性についての法第12条(PCT35条(2))に定める見解、それを裏付ける文献及び説明

1. 見解

新規性(N)	請求の範囲	2, 3, 7-20	有
	請求の範囲	1, 4-6	無
進歩性(I S)	請求の範囲	15-20	有
	請求の範囲	1-14	無
産業上の利用可能性(I A)	請求の範囲	1-20	有
	請求の範囲		無

2. 文献及び説明(PCT規則70.7)

文献1: 若林一敏、外7名、“伝送用LSIを動作合成で開発、機能設計の期間が1/10に短縮”、日経エレクトロニクス、日経BP社、1996.02.12、No.655、P.147-169

文献2: JP 2002-49652 A(安田博)2002.02.15、請求項1-5(ファミリーなし)

文献1に記載のCyberは、動作記述言語BDL(文法はC言語に似ている。)に基づいて高位合成を行いRTL記述を出力している。前記BDLはクロックの変化点(すなわち、クロック境界。)を「\$」により記述することができる。

ここで、\$と\$の間の変数への代入は、変数がどのレジスタにアロケートされるかは不明ではあるものの、レジスタへの代入を生じさせるから、レジスタ代入文と言える。

なお文献1における「=」及び「++」は「固有の演算子」と言える。

また、文献2には第2プログラム記述に相当するものが開示されている。

一方、ある言語で記述されたものを他の言語による記述にコンパイルすることは周知技術である。

してみれば、前記BDLを文献2に記載の記述にコンパイルすることは当業者が適宜なし得ることにすぎない。

Ⅷ. 国際出願に対する意見

請求の範囲、明細書及び図面の明瞭性又は請求の範囲の明細書による十分な裏付についての意見を次に示す。

請求の範囲 1 乃至 2 0 項において、「レジスタ代入文」の技術的範囲が不明確である。

請 求 の 範 囲

1. (補正後) 所定のプログラム言語を流用して記述された第1プログラム記述を回路記述に変換可能なコンパイラであって、

- 5 前記第1プログラム記述は、サイクル精度で回路動作を特定可能とする、固有の演算子が付されたレジスタ代入文とクロック境界記述を含み、
前記回路記述は、前記第1プログラム記述が特定する回路動作を実現するハードウェアを所定のハードウェア記述言語で特定することを特徴とするコンパイラ。

- 10 2. (補正後) 所定のプログラム言語を流用して記述された第1プログラム記述を所定のプログラム言語を用いた第2プログラム記述に変換可能なコンパイラであって、

前記第1プログラム記述は、サイクル精度で回路動作を特定可能とする、固有の演算子が付されたレジスタ代入文とクロック境界記述を含み、

- 15 前記第2プログラム記述は、前のサイクルの状態を参照可能にする為に前記レジスタ代入文を変形した変形代入文と、前記クロック境界記述に対応して前記変形代入文の変数をサイクル変化に伴うレジスタの変化に対応させるレジスタ代入記述挿入文とを含むことを特徴とするコンパイラ。

- 20 3. (補正後) 所定のプログラム言語を流用して記述された第1プログラム記述を、所定のプログラム言語を用いた第2プログラム記述と回路記述に変換可能なコンパイラであって、

前記第1プログラム記述は、サイクル精度で回路動作を特定可能とする、固有の演算子が付されたレジスタ代入文とクロック境界記述を含み、

- 25 前記第2プログラム記述は、前のサイクルの状態を参照可能にする為に前記レジスタ代入文を変形した変形代入文と、前記クロック境界記述

に対応して前記変形代入文の変数をサイクル変化に伴うレジスタの変化に対応させるレジスタ代入記述挿入文とを含み、

前記回路記述は、前記第 2 プログラム記述で定義されるハードウェアを所定のハードウェア記述言語で特定することを特徴とするコンパイラ。

4. 前記所定のプログラム言語は C 言語であることを特徴とする請求項 1 乃至 3 の何れか 1 項記載のコンパイラ。

5. 前記ハードウェア記述言語は R T L レベルの記述言語であることを特徴とする請求項 1 又は 3 記載のコンパイラ。

6. (補正後) タイミング仕様に基づいて回路動作を定義するために、所定のプログラム言語を流用して記述され、サイクル精度で回路動作を特定可能とする、固有の演算子が付されたレジスタ代入文とクロック境界記述を含む第 1 プログラム記述を入力する第 1 処理と、

前記第 1 プログラム記述に基づいて前記タイミング仕様を満足する回路情報を生成する第 2 処理と、を含むことを特徴とする論理回路の設計方法。

7. 前記第 2 処理は、前記第 1 プログラム記述を変換して、レジスタ代入文が入力変数と出力変数を用いて変形されると共に前記クロック境界記述に対応させて前記入力変数を出力変数に代入する記述を含む第 2 プログラム記述を前記回路情報として生成する処理を含むことを特徴とする請求項 6 記載の論理回路の設計方法。

8. 前記第 2 処理は、前記第 2 プログラム記述を変換して、前記タイミング仕様を満足するハードウェアを所定のハードウェア記述言語で特定するための回路記述を更に別の前記回路情報として生成する処理を含むことを特徴とする請求項 7 記載の論理回路の設計方法。

9. 前記プログラム言語は C 言語であることを特徴とする請求項 8 記載

の論理回路の設計方法。

10. 前記第2プログラム記述を用いて設計対象回路のシミュレーションを行う第3処理を更に含むことを特徴とする請求項9記載の論理回路の設計方法。

5 11. 前記第2処理は、前記第1プログラム記述を変換して、前記レジスタ代入文が入力変数と出力変数を用いて変形された記述を含む第2プログラム記述を前記回路情報として生成する処理を含むことを特徴とする請求項6記載の論理回路の設計方法。

10 12. 前記第2処理は、前記第2プログラム記述を変換して、前記クロック境界記述に対応させて前記入力変数を出力変数に代入する記述を含み、所定のプログラム言語で記述されてコンピュータで実行可能な第3プログラム記述を、前記回路情報として生成する処理を含むことを特徴とする請求項11記載の論理回路の設計方法。

15 13. 前記第3プログラム記述を用いて設計対象回路のシミュレーションを行う第3処理を更に含むことを特徴とする請求項12記載の論理回路の設計方法。

20 14. (補正後)タイミング仕様に基づいて回路動作を定義するために、所定のプログラム言語を流用して記述され、サイクル精度で回路動作を特定可能とする、固有の演算子が付されたレジスタ代入文とクロック境界記述を含む第1プログラム記述を入力する入力処理と、

25 前記レジスタ代入文が入力変数と出力変数を用いて変形されると共に前記クロック境界記述に対応させて前記入力変数を出力変数に代入する記述を含み、前記所定のプログラム言語で記述された第2プログラム記述を生成する変換処理と、を含むことを特徴とする論理回路の設計方法。

15. 前記変換処理は、第1プログラム記述に基づいてCFGを生成す